

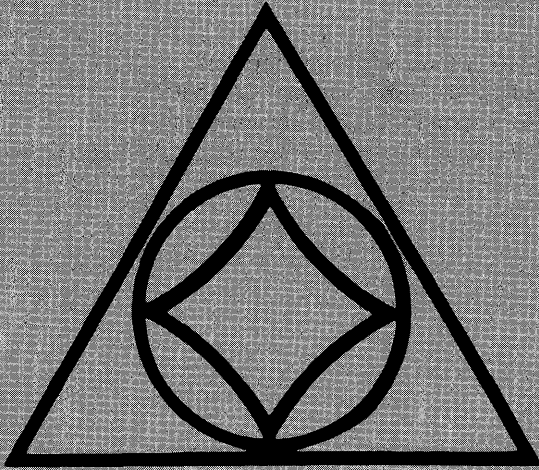
**AFIPS**

**CONFERENCE  
PROCEEDINGS**

**VOLUME 49**

**1980**

**NATIONAL  
COMPUTER  
CONFERENCE**



DON MEDLEY  
Editor and Program Chairman

ELLEN MARIE RANDALL  
Editorial/Production Specialist

HERBERT SAFFORD  
Conference Chairman

**AFIPS PRESS**  
1815 NORTH LYNN STREET  
ARLINGTON, VIRGINIA 22209

# **AFIPS**

## **CONFERENCE PROCEEDINGS**

# **1980**

## **NATIONAL COMPUTER CONFERENCE**

**May 19-22, 1980**  
**Anaheim, California**

The ideas and opinions expressed herein are solely those of the authors and are not necessarily representative of or endorsed by the 1980 National Computer Conference or the American Federation of Information Processing Societies, Inc.

Library of Congress Catalog Card Number 80-66206  
AFIPS PRESS  
1815 North Lynn Street  
Arlington, Va. 22209

© 1980 by AFIPS Press. Copying is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) reference to the AFIPS Proceedings and notice of copyright are included on the first page. The title and abstract may be used without further permission in computer-based and other information-service systems. Permission to republish other excerpts should be obtained from AFIPS Press.

Printed in the United States of America

## Preface



HERBERT B. SAFFORD  
1980 NCC Chairman

The Proceedings of the 1980 National Computer Conference is the most comprehensive review of the current developments in the computing industry. This record of the 1980 NCC Program now becomes a part of the industry's history, and stands as a tribute to Program Chairman Dr. Donald Medley and his committee. With the usability of the computer as an underlying theme of the program, Dr. Medley's committee, the authors whose papers are included in this volume, and many others devoted considerable time and energy during the past year to assure that this program would

be useful and informative to all attendees. They are to be commended for their efforts. Special recognition should be given, also, to the many panelists and speakers who participated in the 1980 NCC Program in a manner that went beyond the formal paper presentations included in these Proceedings. It is my sincere hope that you were able to attend some portion of the program at the 1980 NCC, and that this volume of the Proceedings will be a useful source of information for you for many years to come.

## Introduction



DONALD B. MEDLEY  
1980 NCC Program Chairman

A group of very bright and dedicated computing professionals have labored long and hard to develop a high quality technical program for the 1980 National Computer Conference that demonstrates the dynamics of the computing industry. These Proceedings represent the printed record of the result of over a year's efforts of many, many individuals.

With the usability of the computing tool as the base, the 1980 NCC Program has been oriented to three audiences: the user, the technician and management. Program sessions have been developed for each of these communities within specific topic areas including: the architecture of software and hardware, communications, social impacts, data base management, management support, simulation modeling and image processing, applications considerations and general interest topics. In each of these areas the Program Committee has developed sessions that are educational in nature, sessions that are oriented to the technical details of the topic, sessions that address the management aspects of the topic and sessions for the user of that phase of technology.

A special feature of the 1980 NCC is a group of sessions dealing with the use of the computing tool in the entertainment industry. These sessions describe areas from the generation of music or art to the control of television and movie production. Additionally, in response to the ever growing interest in the use of computers by the non-professional, a

special group of sessions will be presented under the banner of the 1980 NCC Personal Computing Festival, which will cover areas of interest in personal computing. A separate publication will include papers presented in this section of the program.

These Proceedings contain a printed record of the papers presented in the technical program. Summary statements for the many panel sessions were not included due to volume limitations; however, summary statements for each session are published in the conference attendance brochure. The individual area directors have prepared a general summary statement concerning selected groups of sessions and those summaries are in the Proceedings.

The planning and organization of the 1980 NCC Program required the dedicated efforts of many individuals: area directors, session organizers and leaders, panelists and presenters of technical papers, and referees who helped us select the papers to be presented in this volume. I wish to extend my sincere appreciation to all these individuals and, most especially, to the Program Committee. Without their efforts the 1980 NCC Program and these Proceedings would not have been a reality. It is our sincere hope that the program itself proves useful and enjoyable to all those who are able to attend the 1980 NCC and that the Proceedings provide a useful reference source for many years to come.

---

# CONTENTS

Preface .....	iii
Herbert Safford	
Introduction .....	iv
Donald B. Medley	
APPLICATIONS	
A conversational decision support system for resource allocation without explicit objective function .....	1
Fumihiko Mori, Hiroshi Tsuji and Takashi Satō	
Decision support systems: a practical application—Branch office structure .....	7
John R. Wetmiller	
On development tools for small systems: the challenge of economically automating a filing cabinet .....	13
David D. Raber	
A structured information system design for a newspaper organization: a case study .....	23
Mohan R. Tanniru	
SID: a system for interactive design .....	33
Tosiyasu L. Kunii and Minoru Harada	
An overview of a network design system .....	41
W. E. Bracker and B. R. Konsynski	
COMPUTERS AND ENTERTAINMENT	
Area Director Summary .....	49
Suzanne Landa	
A minicomputer system for audio-animatronics show data generation .....	51
Philip C. Stover and R. David Snyder	
Computers and sports: a natural marriage .....	55
Thomas A. Eifler	
Computers helping dance notation help the dance: a vision .....	67
Stephen W. Smoliar	
Automatic Camera Effects System (ACES) .....	73
Steven N. Crane and R. David Snyder	
Automated computer controlled editing sound system (access) .....	83
William R. Deitrick	
The use of computer technology in Magicam slave camera systems .....	87
Dan Slater, Rob King and John Gale	
COMMUNICATIONS	
Area Director Summary .....	91
Kenneth J. Thurber	
Distributed network and multiprocessing minicomputer state-of-the-art capabilities .....	93
Douglas J. Theis	
ARQ performance in SNA networks .....	105
Martin A. Reed and Terence D. Smetanka	

---

Computer communication in NTT remote computing services .....	113
Masatoshi Iwayama and Atsumu Fujiwara	
Local area data distribution .....	121
Thomas G. Albright and Robert J. Wallace	
<b>COMPUTER ARCHITECTURE</b>	
Area Director Summary .....	127
Wesley Chu	
The control data loosely coupled network lower level protocols .....	129
William C. Hohn	
LCN—A loosely coupled network system .....	135
Lowell H. Schiebe	
Derivation and use of a survivability criterion for DDP systems .....	139
Richard E. Merwin and Mohammed Mirhakak	
An operating system kernel mechanism for the poly-processor system PPS-R .....	147
Makoto Amamiya, Naohisa Takahashi, Yutaka Ogawa and Kenji Koyama	
Measures for distributed processing network survivability .....	157
Gene Hilborn	
Architectures for supersystems of the '80s .....	165
Svetlana P. Kartashev and Steven I. Kartashev	
The highly-parallel supercomputers: definitions, applications and predictions .....	181
Hubert H. Love, Jr.	
Database machines and some issues on DBMS standard .....	191
Stanley Y. W. Su, Hsu Chang, George Copeland, Paul Fisher, Eugene Lowenthal and Stewart Schuster	
CONLAN—A formal construction method for hardware description languages: basic principles .....	209
Robert Piloty, Mario Barbacci, Dominique Borrione, Donald Dietmeyer, Fredrick Hill and Patrick Skelly	
CONLAN—A formal construction method for hardware description languages: language derivation .....	219
Robert Piloty, Mario Barbacci, Dominique Borrione, Donald Dietmeyer, Fredrick Hill and Patrick Skelly	
CONLAN—A formal construction method for hardware description languages: language application .....	229
Robert Piloty, Mario Barbacci, Dominique Borrione, Donald Dietmeyer, Fredrick Hill, Patrick Skelly	
Design decisions for the intelligent database machine .....	237
Robert Epstein and Paula Hawthorn	
DIALOG—A distributed processor organization of database machine .....	243
Benjamin W. Wah and S. Bing Yao	
<b>DATA BASE MANAGEMENT</b>	
Area Director Summary .....	255
Alyce Jackson	
System deadlocks resolution .....	257
Koji Nezu	
Database semantic integrity for a network data manager .....	261
Elizabeth Fong and Stephen R. Kimbleton	
Concurrency coordination in a locally distributed database system .....	269
Gruia-Catalin Roman	
An introduction to computed chaining .....	275
Kuo-Chung Tai and Alan L. Tharp	

---

A federated architecture for database systems .....	283
Dennis McLeod and Dennis Heimbigner	
Area Director Summary .....	291
Linda Taylor	
Definition of database transactions by the casual user .....	293
Fred J. Maryanski and C. Steven Roush	
Programming with data frames for everyday data items .....	301
David W. Embley	
Implementing data management .....	307
Daniel S. Appleton	
Area Director Summary .....	317
Vincent Lum	
Properties of relationships and their representation .....	319
Ramez El-Masri and Gio Wiederhold	
<b>EARTH RESOURCES</b>	
Area Director Summary .....	327
Leigh Rosenberg	
BALDR-1: a solar thermal system simulation .....	329
Joseph G. Finegold and F. Ann Herlevich	
Overview of the alternative power system economic analysis model .....	335
Richard B. Davis and Jerome V. V. Kasper	
Computer simulation of the operations of utility grid connected photovoltaic power plants .....	341
Chester S. Borden	
Computer simulation of solar electric generating plants in a utility grid .....	347
S. Young, O. Merrill, R. Knowles and Y. Gupta	
Area Director Summary .....	355
Roger Firestone	
Numerical algorithms for parallel computers .....	357
David K. Stevenson	
Design of an interactive matrix calculator .....	363
Cleve Moler	
<b>IMAGE PROCESSING</b>	
Area Director Summary .....	369
Andrew Tescher	
Derivation of invariant scene characteristics from images .....	371
Berthold K. P. Horn	
Image understanding architectures .....	377
Graham R. Nudd	
Map-guided interpretation of remotely-sensed imagery .....	391
J. M. Tenenbaum, H. G. Barrow, R. C. Bolles, M. A. Fischler and H. C. Wolf	
CCITT standardization for digital facsimile .....	409
T. L. McCullough	
The application of optical character recognition techniques to bandwidth compression of facsimile data .....	415
Patrice J. Capitant and Robert H. Wallis	



---

Facsimile image coding .....	423
Joan L. Mitchell	
<b>MEDICAL IMAGING AND EDUCATION</b>	
Description and evaluation of a system for high-speed, three-dimensional computed tomography of the body: the dynamic spatial reconstructor .....	427
Richard A. Robb and Barry K. Gilbert	
3-D viewer for interpretation of multiple scan sections .....	437
Brent Baxter	
Absolute limits on image processing .....	441
David G. Brown, Robert F. Wagner and Mary Pastel Anderson	
Generalized methodology for the comparison of diagnostic imaging instrumentation .....	445
Leon Kaufman and Dale Shosa	
<b>MANAGEMENT</b>	
Balancing processor shares of scheduling classes through controlled allocation of memory .....	453
K. V. Sastry	
Area Director Summary .....	457
John C. Biddle	
Applications of exemplary programming .....	459
William S. Faught	
Multiprocessor software engineering training: a case study .....	465
Christine L. Braun	
Development of a microprocessor support facility for large organizations .....	473
Bruce E. Stock and Miguel A. Ulloa	
Future management concerns regarding office automation .....	479
Gary D. Beamer	
<b>OFFICE AUTOMATION</b>	
Area Director Summary .....	483
Walter E. Ulrich	
Introduction to electronic mail .....	485
Walter E. Ulrich	
Implementation considerations in electronic mail .....	489
Walter E. Ulrich	
Experiences of an electronic mail vendor .....	493
Jeffrey B. Holden	
Electronic message system as a function in the integrated electronic office .....	499
Harold E. O'Kelley	
The growing use of electronic mail by airlines .....	503
James C. Goodlett	
Metamorphosis: facsimile communications, electronic mail and office productivity .....	509
John E. Cochran	
Texas Instruments computer communication network and its support for the automated office .....	515
John W. White	
Implementing electronic mail in a telephone system: more than just talk .....	527
Gerald Tomanek	

---

An office form flow model .....	533
Ivor Ladd and D. C. Tsichritzis	
Design principles of an office specification language .....	541
Michael Hammer and Jay S. Kunin	
Automated workflow control: a key to office productivity .....	549
L. S. Baumann and R. D. Coop	
Streamlining office procedures—An analysis using the information control net model .....	555
Carolyn L. Cook	
Area Director Summary .....	567
James Carlisle	
Provisions for flexibility in the Linköping office information system (LOIS) .....	569
Erik Sandewall, Göran Hektor, Anders Ström, Claes Strömberg, Ola Strömfors, Henrik Sörensen and Jaak Urmi	
<b>SECURITY AND PRIVACY</b>	
Area Director Summary .....	579
Rein Turn	
Privacy protection and transborder data flows .....	581
Rein Turn	
Transborder data flow: legal persons in privacy protection legislation .....	587
Susan H. Nycum and Susan Courtney-Saunders	
<b>SIMULATION</b>	
Area Director Summary .....	595
Lance A. Leventhal	
Using preliminary Ada in a process control application .....	597
M. E. Gordon and W. B. Robinson	
Computer aided heat penetration tests for the food canning industry .....	607
Paul Sagues	
A cross-impact simulation forecast of the data processing industry .....	613
Paul Herbert Rosenthal	
Organization of the TRAC processor-memory subsystem .....	623
R. N. Kapur, U. V. Premkumar and G. J. Lipovski	
An overview of the Texas Reconfigurable Array Computer .....	631
Matthew C. Sejnowski, Edwin T. Upchurch, Rajan N. Kapur, Daniel P. S. Charlu, and G. Jack Lipovski	
Design and implementation of the banyan interconnection network in TRAC .....	643
U. V. Premkumar, R. Kapur, M. Malek, G. J. Lipovski and P. Horne	
<b>SOCIAL IMPACT</b>	
The advent of trusted computer operating systems .....	655
Stephen T. Walker	
<b>SOFTWARE MANAGEMENT</b>	
Area Director Summary .....	667
Donald Reifer	

---

**SOFTWARE ENGINEERING TECHNOLOGY TRANSFER**

Area Director Summary .....	669
Lorraine Duvall	
An integrated support software network using NSW technology .....	671
Richard A. Robinson and Emily A. Krzysiak	
The role of an information analysis center in software engineering technology transfer .....	677
Jon Martens and Lorraine Duvall	
Considerations in the transfer of software engineering technology .....	683
Michael J. McGill	

**SOFTWARE TOOLS AND TECHNIQUES**

A technique for comparative assessment of software development management policies .....	687
Brendan D. L. Mulhall and Steven M. Jacobs	

**SOFTWARE RELIABILITY**

Area Director Summary .....	695
Herbert Hecht	
Standard error classification to support software reliability assessment .....	697
John B. Bowen	
What makes a reliable program—few bugs, or a small failure rate? .....	707
B. Littlewood	
Software reliability and advanced avionics .....	715
Gerard E. Migneault	

**SOFTWARE LANGUAGES**

Area Director Summary .....	721
Russell J. Abbott	
A linguistic comparison of MUMPS and COBOL .....	723
Thomas Munnecke	
The design of PLAIN—Support for systematic programming .....	731
Anthony I. Wasserman	
Some practical experiences with the Pascal language .....	741
G. G. Gustafson, T. A. Johnson and G. S. Key	
UCSD Pascal <sup>TM</sup> : a portable software environment for small computers .....	747
Mark Overgaard	

**SOFTWARE QUALITY ASSURANCE**

Area Director Summary .....	755
Kurt F. Fischer	
Measuring program complexity in a COBOL environment .....	757
Jean Zolnowski and Dick B. Simmons	
The complexity of an individual program .....	767
John L. McTap	
An information theory based complexity measure .....	773
Eli Berlinger	

SOFTWARE ENGINEERING EDUCATION

Area Director Summary ..... 781  
 Barry Boehm

SPECIAL TOPICS

Area Director Summary ..... 783  
 Gene Smith

Technology development, severed ventures, and other aspects of corporate venture capital ..... 785  
 Jean E. de Valpine

Recommendations for increasing the availability of capital ..... 791  
 Richard C. Pflager

Corporate venture capital in the computer industry ..... 795  
 Kenneth W. Rind and Gene I. Miller

Structured procedure for comparison and selection of computer system designs ..... 801  
 Antonio Vallone

PAPER FAIRE

Extracting unique rows of a bounded degree array using tries ..... 807  
 Douglas Comer

A look at making the ADP procurement process more efficient—Temporary regulation 46 ..... 811  
 Roger J. Gorg, George N. Baird and Judith A. Parks

An information base for procedure independent design of information systems ..... 817  
 Levent Ormancioglu

Comparing load & go and link/load compiler organizations ..... 823  
 William L. Wilder

A link between polygon and grid representations of land resource information systems ..... 827  
 Devon Nickerson

Risk analysis in the 1980's ..... 831  
 Jerome Lobel

A mathematical model of character string manipulation ..... 837  
 Sakti Pramanik

Policy, values and EFT research: anatomy of a research agenda ..... 841  
 Kenneth L. Kraemer and Kent W. Colton

A linear programming model for optimal computer network protocol design ..... 855  
 John F. Heafner and Frances H. Nielsen

Extracting service features from protocol documents ..... 863  
 John F. Heafner, Frances H. Nielsen and M. Wayne Shiveley

Verification of information in a file ..... 871  
 Jainendra K. Navlakha

Translating non-standard extensions to standard Pascal ..... 877  
 Viswanathan Santhanam

The flexible console—FLEXICON ..... 883  
 David L. Steinberg

The INTEL® 8087 numeric data processor ..... 887  
 John F. Palmer

Home computing—A vision in search of a marketplace: areas of needed research .....	895
John E. Ruchinkas, Charles W. Steinfield and Lynne L. Svenning	
1980 National Computer Conference Committees .....	903
NCC '80 Area Directors .....	905
NCC '80 Session Chairmen .....	906
NCC '80 Referees .....	909
NCC '80 Speakers and Panelists .....	911
American Federation of Information Processing Societies .....	914
Author Index .....	918

# A conversational decision support system for resource allocation without explicit objective function

by FUMIHIKO MORI, HIROSHI TSUJI and TAKASHI SATO

*Hitachi, Ltd.*  
Ohzenji, Kawasaki, Japan

## INTRODUCTION

This paper presents a conversational multiobjective decision support system. The system is called RAINBOW: Resource Allocation in Business Operation under Uncertain Worth. Our focus in the design and development of RAINBOW is placed on the loosely structured decision situation where the objective functions are given only implicitly and, as such, should be approximated by the decision maker as the decision making process proceeds.

In the following, we first describe the specific decision problem treated here. This is the loans budgeting decision in a bank in Japan. Next we will see that the budget allocation procedure which had been employed in the bank prior to the design and implementation of RAINBOW can be represented by a set of simple linear equations. Basically RAINBOW supports the process of convergence to a preferred alternative budget plan by giving the decision maker helpful information for him to form consistent evaluations of the utility function, the objective functions and the solution for the budget allocation plan. Description of the functions of RAINBOW will be the main part of this paper. Lastly we will comment on RAINBOW in the framework of the multiple objective programming.

## AN ILLUSTRATIVE DECISION PROBLEM

Direct motivation for the development of RAINBOW came from the problem of allocating the loans budget to branch offices of a bank, although RAINBOW is applicable to general resource allocation problems.

This section describes the specific problem of loan budgeting as a rationale for the functional design of RAINBOW.

Executives in the loan department of a bank annually or biannually allocate loan budget to each of its branch offices. The planning is a routine decision making task, but it is a difficult and time consuming problem. In many cases the total amount of the budget is determined by the total deposit in the bank and financial policy of the government at the time. Therefore the total budget is usually not a decision variable but a constraint on the upper bound of the sum of the allocatable resources.

If we are to formulate the problem as an optimization problem, we will have a mathematical programming problem of the following form:

$$\text{maximize } u = f(u_1, u_2, \dots, u_N) \quad (1)$$

$$\text{s.t. } u_1 + u_2 + \dots + u_N \leq S \quad (2)$$

$$u_i \geq 0, \text{ for } i = 1, \dots, N \quad (3)$$

where

$N$  = number of branch offices,

$S$  = upper bound on the sum of budgets,

$u_i$  = budget allocated to the  $i$ th branch,

$u$  = expected utility as a function of budget plan,  $(u_1, u_2, \dots, u_N)$ .

At first we considered an approach by some mathematical programming model such as above. But, through extensive interviews with the managers in the loan department of the bank, it became clear that explicit identification of a suitable objective function was a difficult and thankless task.

The fact that, for instance, one dollar allocated to branch  $i$  and  $j$ , respectively, will yield the same return on investment, provided that such factors as average interest rate, average operation cost per transaction and risk are the same in both branches, would illustrate the difficulties in the model formulation in the form of equation (1). Although these factors do differ among the branches, corporate-strategic considerations other than these quantifiable factors play a larger part in budget allocation. For example, a branch with high operation cost ratio may be a newly opened one and the policy of the bank may be to expand its territory, in spite of its relatively high cost at present, by aggressive operation in the financial market of the particular locality. A branch in a densely inhabited district may not need a big budget in spite of its large amount of deposits, since its expected role is collection of deposits from the households in its territory. The necessity to take account of these strategic considerations is the rule rather than the exception.

The budgeting procedure, then, depends on a number of factors which are difficult to quantify. The factors involved include, among others, the geographical and strategic lo-

cation of the branch and its position and expected role in the overall corporate strategy of the bank.

The kind of decision situation described above occurs quite often in daily operations of any business and other organizations. It requires many subjective and intuitive judgments as to the utility gained from each of the alternative actions. Evaluations of other factors relevant to the problem are also bound to be subjective to a certain degree.

An approach we can take in these decision situations is to try to help the thinking process of the decision maker.

An important function of the budget planning is to express the policy of the organization and to get the consensus of the people involved in order to assure well-coordinated organizational activity. Then, an important requirement in the development of a decision support system is to help the decision maker to express his views, judgments and evaluations, albeit subjective or intuitive, and to make them something objective in that they have been expressed and now are the objects of discussion, criticism and modification. It will be our concern in this loosely structured decision situation to facilitate this dialectical process of subjectivity and objectivity.

Many researchers in the field of multiobjective decision making have been working on this standpoint. The Surrogate Worth Tradeoff (SWT) method by Haines and Hall<sup>1,2</sup> is effective when the decision makers can evaluate the marginal tradeoffs among objective functions. Multiattribute Utility Theory (MAUT) by Keeney and Raiffa<sup>3</sup> gives a method of constructing a utility function based on preference structure and indifferent curves of each pair of objectives. These methods have been applied to large-scale regional development planning.<sup>4,5</sup> Geoffrion et al.<sup>6</sup> proposed a method of conversational decision support system based on the Frank-Wolfe algorithm.

Many of those multiobjective decision making systems, of which only very few are mentioned above, depend heavily on the tradeoff concept. Decision makers who want to use either the SWT method or MAUT method must be ready to evaluate the tradeoffs among conflicting objectives explicitly and quantitatively. It is a task of system designer to make it easy for the decision maker to articulate the tradeoff evaluations.

## ANALYSIS OF BUDGET DECISION MAKING PROCESS

As mentioned in the previous section, it became clear that formulation of the allocation problem in a mathematical programming problem with scalar objective function as in equations (1)-(3) was not suited to this case. The problem is multiobjective. Generating noninferior solutions was not possible since managers in the loan department felt they had trouble in identification and quantification of the objectives. Then, as the first step of the system design, efforts were made to model the actual budget allocation procedure.

Interviews with the managers revealed that the budget allocation procedure actually taken in the bank could be summarized as follows.

### Step 1

Managers select "evaluation elements" or "allocation elements." These are items for the evaluation of the branch offices. Typical examples of the element are "basic evaluation," "branch characteristics," "growth potential" and "strategic consideration."

### Step 2

The total budget  $S$  is allocated to the elements at first. If the managers have selected  $m$  elements  $E_1, E_2, \dots, E_m$  in Step 1 above, and give weight  $e_j$  to the element  $E_j$ , then  $E_j$  receives  $e_j S$  out of the total  $S$ . The weights should satisfy

$$e_1 + e_2 + \dots + e_m = 1.0 \quad (4)$$

$$e_j > 0, \text{ for } j = 1, \dots, m. \quad (5)$$

### Step 3

Managers evaluate branches with respect to each of the evaluation elements. Let  $B_1, B_2, \dots, B_N$  denote  $N$  branches, and let  $a_{ij}$  denote the numerical evaluation of the branch  $B_i$  with respect to the element  $E_j$ . According to these evaluations branch  $B_i$  receives  $a_{ij} e_j S$  out of the amount  $e_j S$  allocated to the evaluation element  $E_j$  in Step 2. Since negative value of  $a_{ij}$  can result in negative allocation to  $B_i$  when  $e_j$  is quite large,  $a_{ij}$ s are assumed to satisfy the nonnegativity condition:

$$a_{ij} \geq 0, \text{ for } i = 1, \dots, N \text{ and } j = 1, \dots, m. \quad (6)$$

Also

$$a_{1j} + a_{2j} + \dots + a_{Nj} = 1.0 \quad (7)$$

### Step 4

The budget allocated to branch  $B_i$  is just the sum of  $a_{ij} e_j S$  over  $j = 1, \dots, m$ . Let  $u_i$  denote the budget for  $B_i$ , then

$$u_i = a_{i1} e_1 S + a_{i2} e_2 S + \dots + a_{im} e_m S, \quad (8)$$

for  $i = 1, \dots, N$ .

In matrix notation, equation (8) is

$$\underline{u} = A \underline{e} S \quad (9)$$

where

$$\underline{u} = (u_1, u_2, \dots, u_N)^t \quad (10)$$

$$\underline{e} = (e_1, e_2, \dots, e_m)^t \quad (11)$$

$$A = (\underline{a}_1, \underline{a}_2, \dots, \underline{a}_m) \quad (12)$$

$$\underline{a}_i = (a_{i1}, a_{i2}, \dots, a_{in})^t, \text{ for } i = 1, \dots, m \quad (13)$$

with  $t$  denoting the transpose. The procedure represented in equation (9) is illustrated in Figure 1.

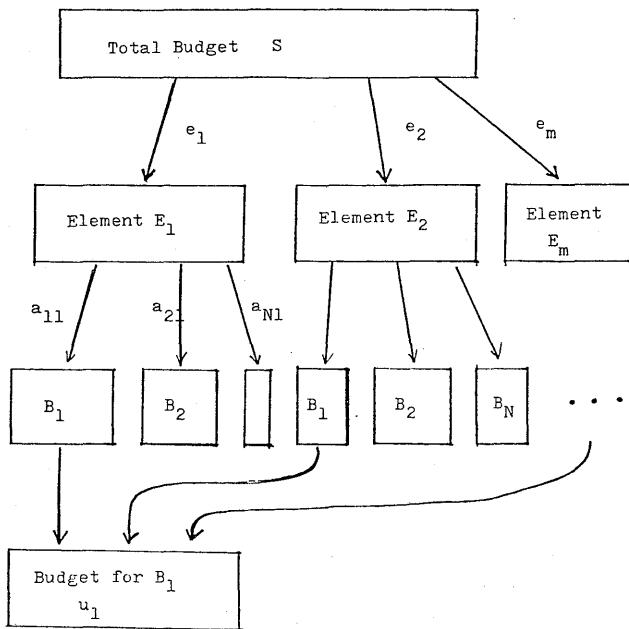


Figure 1—Linear allocation model.

**RAINBOW: A CONVERSATIONAL DECISION SUPPORT SYSTEM**

Basically speaking, when he uses the allocation method of equation (9), the decision maker has only to select a set of evaluation elements  $\{E_1, E_2, \dots, E_m\}$  and determine the values of the weight vector  $e$  and the evaluation matrix  $A$ . But the problems of (1) selection of the evaluation elements, (2) determination of suitable weights for the elements, and (3) evaluation of the branch offices with respect to each of the evaluation elements, all required considerable time and efforts.

Managers felt that they might not have identified all of the significant evaluation elements. They wanted to try various alternative sets of evaluation elements, but the number of branches, in this case more than 100, prohibited them from trying more than a few sets of elements. The weights, even for the same set of elements, should change according to the changes in social and business environment; but again they were in need of tools for finding adequate values for the weights. Evaluation of the branches was also a tedious and difficult task.

Discussion with the managers revealed the general view that these problems were not to be solved uniquely by some set of fixed criteria, but to be solved through an iteration of trial-and-error simulation, out of which an admissible solution was expected to emerge. Then, the support system should help the decision maker to observe the change in the allocation plan corresponding to the changes in the set of evaluation elements, modifications of the element weights and re-evaluations of the branches.

Another need which the results of the requirement analysis (the SADT: Structural Analysis and Design Technique<sup>7</sup> was

used) revealed was the inverse computation of the element weight vector  $e$  and the branch evaluation matrix  $A$ . In many cases the managers have, or they think they have, an *a priori* solution to the question, "What is the best allocation plan?" Similarly the decision maker has his idea as to the most appropriate values of  $e$  and  $A$ . But, without any reference values, i.e., tentative values of  $e$  and  $A$ , it is often difficult for him to express it quantitatively. If a trial value is given to him, it is relatively easy for him to specify how it should be modified.

Let these "intuitive solutions" be denoted as  $\tilde{u}, \tilde{e}$  and  $\tilde{A}$ . It can be said that a necessary condition for the decision making process of budget allocation to come to an end is that these intuitive solutions, after undergoing the articulation-modification process, have become consistent in the sense that they satisfy:

$$\tilde{u} = \tilde{A}eS. \tag{14}$$

This consistency, or agreement, among the intuitive values of  $u, e$  and  $A$  is hard to attain. One reason for this is that, as already mentioned, the number of branches is quite large. Another reason is that the managers did not have a means of finding values of  $e$  and/or  $A$  corresponding to the allocation plan  $u$  which they could specify.

In order to check the consistency condition of equation (14) and to facilitate the convergence to the set of values  $\tilde{u}, \tilde{e}$  and  $\tilde{A}$  which satisfy equation (14), the decision support system should have the capability of inverse computation of  $e$  and  $A$  corresponding to the allocation plan  $u$  given by the decision maker. On these grounds RAINBOW was designed as an on-line conversational system based on the three basic algorithms described below.

*Algorithm 1*

This is the direct application of equation (9), i.e., computation of the allocation vector  $u$  when values of the branch evaluation matrix  $A$  and the element weight vector  $e$  are specified by the decision maker. Schematically this is

$$e, A \text{ and } S \rightarrow u.$$

Decision maker can use Algorithm 1 to analyze sensitivity of the allocation plan to changes in the values of  $e$  and  $A$ .

*Algorithm 2*

This algorithm computes weight vector  $e$  corresponding to the specified values of  $u, A$  and  $S$ . Schematically this is

$$u, A \text{ and } S \rightarrow e.$$

That is, the decision maker can use the Algorithm 2 to obtain answer to the question, "If I allocate the budget to branches like this value of  $u$  vector, what value does it mean I am assigning to the weight vector  $e$ ? If that value of  $e$  does not deviate much from the value I can accept, I might be able to say that my evaluations of the three decision variables,



$\underline{u}$ ,  $\underline{e}$  and  $A$ , are in agreement with each other and I am in business. If it does not, I have something further to think about."

The least-squares solution for  $\underline{e}$  is given by

$$\begin{bmatrix} e_1 \\ e_2 \\ e_m \\ \lambda/2 \end{bmatrix} = \begin{bmatrix} (\underline{a}_1, \underline{a}_1) & (\underline{a}_1, \underline{a}_2) & \dots & (\underline{a}_1, \underline{a}_m) & 1 \\ (\underline{a}_2, \underline{a}_1) & (\underline{a}_2, \underline{a}_2) & \dots & (\underline{a}_2, \underline{a}_m) & 1 \\ \dots & \dots & \dots & \dots & \dots \\ (\underline{a}_m, \underline{a}_1) & (\underline{a}_m, \underline{a}_2) & \dots & (\underline{a}_m, \underline{a}_m) & 1 \\ 1 & 1 & \dots & 1 & 0 \end{bmatrix} \begin{bmatrix} (\underline{a}_1, \underline{u})/S \\ (\underline{a}_2, \underline{u})/S \\ \dots \\ (\underline{a}_m, \underline{u})/S \\ 1 \end{bmatrix} \quad (15)$$

where  $(\cdot)$  denotes the inner product and  $\lambda$  is the Lagrange multiplier for the constraint (4). Here we note that, in general,  $N \gg m$ , and the use of the least-squares estimation is justified in the statistical sense.

The weight vector  $\underline{e}$  obtained from equation (15) does not necessarily satisfy the positivity condition (5). If the estimated weight  $e_j$  for some element  $E_j$  is nonpositive, it indicates strong inconsistency between the specified values of  $\underline{u}$  and  $A$ . Also the inverse matrix in the equation (15) does not exist when vectors  $\underline{a}_j$  in matrix  $A$  are linearly dependent. This means that the set of elements  $\{E_1, \dots, E_m\}$  has redundancy in the sense that evaluation vector  $\underline{a}_j$  of certain element  $E_j$  can be represented as a combination of evaluation vectors for other elements. RAINBOW gives warning messages when the nonpositivity or the redundancy phenomena occurs.

There can be a case where a decision maker does not want to eliminate the redundancy in the evaluation elements; or the least-squares method gets ill-posed and cannot find positive solution for  $\underline{e}$  vector when only a slight change in  $\underline{u}$ ,  $\underline{e}$  or  $A$  would be sufficient to give positive  $e$ . Then the decision maker can obtain estimation of  $\underline{e}$  which satisfies both conditions (4) and (5) by solving the following nonlinear programming problem:

$$\text{minimize } \|\underline{u} - A\underline{e}S\| \quad (16)$$

$$\text{s.t. } e_1 + e_2 + \dots + e_m = 1.0 \quad (17)$$

$$e_j > 0, \text{ for } j = 1, \dots, m, \quad (18)$$

where  $\|\cdot\|$  denotes Euclidean norm of vectors. A random search method, called the hyperconical random search,<sup>8</sup> is used to avoid the numerical difficulty which may be caused by the linear dependency.

### Algorithm 3

Since it is not possible to estimate all elements of the matrix  $A$  at one time, this algorithm performs the task of inverse computation in a limited way, schematically shown as

$$\underline{u}, \underline{e}, A^{(j)}, \text{ and } S \rightarrow \underline{a}_j,$$

where  $A^{(j)}$  denotes the submatrix of  $A$  formed by deleting  $\underline{a}_j$  from  $A$ . That is, we assume that only the evaluation vector  $\underline{a}_j$  with respect to evaluation element  $E_j$  is unknown and

compute  $\underline{a}_j$  uniquely using the following equation:

$$\underline{a}_j = (\underline{u}/S - \sum a_{ik}e_k)/e_j \quad (19)$$

for  $i=1, \dots, N$ . Here the summation is taken for  $k=1, \dots, m$  except for  $k=j$ . That is, the decision maker can obtain an answer to the question, "If I allocate the budget like in this vector  $\underline{u}$ , and evaluate the elements like in this vector  $\underline{e}$ , and if I am uncertain as to the appropriateness of my present evaluation of branches with respect to this particular evaluation element  $E_j$ , then what is the  $\underline{a}_j$  corresponding to my  $\underline{u}$  and  $\underline{e}$ , provided that, of course, my evaluations for the other elements are tentatively considered appropriate? If I can accept the value for  $\underline{a}_j$ , I am in a good shape; if not I must think further."

Again, when there is strong inconsistency among the specified values of  $A^{(j)}$ ,  $\underline{e}$  and  $\underline{u}$ ,  $\underline{a}_j$ s of equation (19) do not necessarily satisfy the nonnegativity condition (6). A warning message is given in this case also.

The following is a scenario of the decision making process in which RAINBOW is used to select a budget plan.

#### Step 1

Decision maker selects evaluation elements tentatively, evaluates them by assigning weights he considers suitable and evaluates branch offices with respect to each of the elements.

#### Step 2

Decision maker allocates the total budget to branches using the Algorithm 1.

#### Step 3

Decision maker checks the tentative budget plan displayed on the terminal. If he feels that it is appropriate and he can accept it, he terminates the session. If he feels that it should be modified, or that he should try other values of  $\underline{e}$  and/or  $A$ , he goes to Step 4.

#### Step 4

Decision maker determines what should be modified: (1) if the decision maker feels that the evaluation elements  $\{E_1, \dots, E_m\}$  should be changed, he goes to Step 1; (2) if he feels that the element weights should be re-evaluated, he goes to Step 5; (3) if he feels that the branches should be re-evaluated, he goes to Step 6; (4) if he feels that both the elements and the branch offices should be re-evaluated, he goes to Step 1.

### Step 5

Decision maker selects one of the following alternatives: (1) if he wants to modify the element evaluation, he specifies new  $\underline{e}$  and goes to Step 2, or (2) if he wants to modify the budget  $\underline{u}$  and observe the corresponding element weight  $\underline{e}$ , he gives new  $\underline{u}$ , compute  $\underline{e}$  using the Algorithm 2, and goes to Step 7.

### Step 6

Decision maker selects one of the following alternatives: (1) if he wants to modify the branch evaluation matrix  $A$ , he specifies new  $A$  and goes to Step 2, or (2) if he wants to modify the budget  $\underline{u}$  and observe the corresponding branch evaluation, he first selects  $\underline{a}_j$  which should be computed, computes  $\underline{a}_j$  using the Algorithm 3, and goes to Step 8.

### Step 7

If the decision maker feels that the element weight vector  $\underline{e}$  computed by the Algorithm 2 is appropriate, he goes to Step 2, otherwise he goes to Step 5.

### Step 8

If the decision maker feels that the branch evaluation vector computed by the Algorithm 3 is appropriate, he goes to Step 3, otherwise he goes to Step 6.

The above scenario is only an example of a session with RAINBOW. Users of RAINBOW can use any of the three algorithms at any stage of the session.

## DISCUSSION AND CONCLUDING REMARKS

In this final section we review the decision support system RAINBOW in the general framework of multiple objective programming.

A general form of multiobjective programming problem based on the utility concept is:

$$\underset{\underline{x} \in X}{\text{maximize}} \quad U[g_1(\underline{x}), g_2(\underline{x}), \dots, g_m(\underline{x})] \quad (20)$$

where  $\underline{x} = (x_1, \dots, x_N)'$  and  $X$  is a subset of  $R^N$ . Functions  $g_j$  are objective functions and  $U$  gives the utility as a function of  $G, s$ .

If both  $U$  and  $g_j$ s are explicitly known, the gradient vector of  $U$  with respect to  $\underline{x}$  is:

$$\nabla_{\underline{x}} U = \sum_{j=1}^m \left( \frac{\partial U}{\partial g_j} \right) \nabla_{\underline{x}} g_j. \quad (21)$$

Geoffrion et al.<sup>6</sup> presented a conversational decision support

method for the case where  $g_j$ s are explicitly given but  $U$  is known only implicitly. Their method is called IFW (Interactive Frank-Wolfe) method and employs the Frank-Wolfe algorithm to solve the direction problem which is specified when the underlying utility function  $U$  is locally approximated by the decision maker.

As indicated in this paper, there are cases in the real-world decision situation in which not only the underlying utility function  $U$  but the objective functions  $g_j$  are not given explicitly and must be locally approximated as the decision making process proceeds. The Multiattribute Utility Theory is an attempt to cope with such situations.

Since linear approximation is one of the most elementary methods of function approximation, let us assume that the decision maker can supply a local linear approximation

$$g_j(x) = a_{1j}x_1 + a_{2j}x_2 + \dots + a_{Nj}x_N + b_j \quad (22)$$

for each of the "true"  $g_j$ s in the neighborhood of any nominal solution  $\underline{x}$  in  $X$ . Here the coefficients  $a_{ij}$  depend on the incumbent solution  $\underline{x}$  and therefore they should be locally re-estimated in the iterative process of decision making.

When the objective functions  $g_j$  are given in their linearly approximated form of equation (22), equation (21) is written in the form of matrix-vector multiplication as

$$\nabla_{\underline{x}} U = \nabla_{\underline{x}} g \nabla_{\underline{g}} U \quad (23)$$

where  $g = (g_1, \dots, g_m)'$  and

$$\nabla_{\underline{g}} U = \left( \frac{\partial U}{\partial g_1}, \frac{\partial U}{\partial g_2}, \dots, \frac{\partial U}{\partial g_m} \right) \quad (24)$$

Also,  $\nabla_{\underline{x}} g$  is just the  $A$  matrix given in equation (12).

It is not difficult to notice the correspondence between the evaluation elements in the allocation procedure and the "attributes," or objective functions, in the theory of multiple objective decision making. Then the correspondence between equation (9) and (23) is also clear. Since  $\nabla_{\underline{g}} U$  represents sensitivities of  $U$  to the changes in the objective functions  $g_j$ , it corresponds to the element weight vector  $\underline{e}$ . The correspondence between the branch evaluation matrix  $A$  and the gradient matrix  $\nabla_{\underline{x}} g$  is apparent when we consider  $\underline{x}$  as the allocation vector  $\underline{u}$ .

From these observations we know that equation (9) is actually the gradient  $\nabla_{\underline{x}} U$  when objective functions  $g_j$  are given in linearly approximated form. Equation (9) means that, under the constraint of (2), the total fund  $S$  should be allocated exactly proportionately to the ratio of the elements of the gradient vector.

Then, RAINBOW is a decision support system for the case where not only the underlying utility function  $U$  but the objective functions  $g_j$  cannot be given except in the local approximations. RAINBOW helps the decision maker to estimate the two gradients  $\nabla_{\underline{x}} g$  and  $\nabla_{\underline{g}} U$  in a trial-and-error manner.

Since we assumed that  $g_j$ s are only implicitly known we have lost the assurance of local optimality we can expect when we locally fix an evaluation of the utility function. This is the price to be paid for loosening the assumptions as to the type of decision situations to be covered. In place of the algorithm for solving the local optimization problem, RAINBOW is equipped with the three algorithms to help the decision maker to form the consistency among his answers to the question, "What are the most appropriate values for  $\underline{g}$ ,  $\underline{u}$  and  $A$ ?"

Clearly this is not the only means of decision support in these loosely structured situations. Other approaches must be experimented.

As our concluding remark we want to point out that the development of RAINBOW was not initiated by the theoretical considerations discussed above; it was based on the analysis of the actual behavior, or procedure, of the decision makers in the bank. It seems that the importance of the behavioral analysis of the actual decision procedure increases when we are to design a decision support system under the loosely structured situation.

## REFERENCES

1. Haimes, Y. Y. and Hall, W. A., "Multiobjectives in Water Resource Systems Analysis: The Surrogate Worth Trade-off Method," *Water Resources Research*, vol. 10 no. 4, 1974, pp. 615-624.
2. Hall, W. A. and Haimes, Y. Y., "The Surrogate Worth Trade-off Method with Multiple Decision Makers," in *Multiple Criteria Decision-Making: Kyoto 1975*, M. Zelney (ed.), Springer-Verlag, New York, 1976, pp. 207-233.
3. Keeney, R. L. and Raiffa, H., *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, J. Wiley, New York, 1976.
4. Haimes, Y. Y., Das, P., and Sung, K., "Multiobjective Analysis and Related Land Resources Planning," in *Water Resources and Land Use Planning*, P. Laconte and Y. Y. Haimes (eds.), 1979.
5. de Neufville, R. and Keeney, R. L., "Use of Decision Analysis in Airport Development for Mexico City," in *Analysis of Public Systems*, A. W. Drake, R. L. Keeney and P. M. Morse (eds.), M.I.T. Press, 1972.
6. Geoffrion, A. M., Dyer, J. S., and Feinberg, A., "An Interactive Approach for Multi-criterion Optimization, with an Application to the Operation of an Academic Department," *Management Science*, 19(4), 1972, pp. 357-368.
7. Dickover, M. E., McGowan, C. L., and Ross, D. T., "Software Design using SADT," in *Proceedings of the ACM National Conference*, Seattle, Washington, October 1977.
8. Wozny, M. J. and Heydt, G. T., "Hyperconical Random Search," *Journal of Dynamic Systems, Measurement and Control*, March 1972, pp. 71-77.

# Decision support systems: a practical application—Branch office structure

by JOHN R. WETMILLER

Digital Equipment Corporation  
Maynard, Massachusetts

## INTRODUCTION

As the decade of the 1980s approaches, it is generally agreed that computer companies will stress not technology, but rather customer service functions as a means to differentiate themselves from one another. Consequently, the need to understand the customer's service requirements and to plan for them is tantamount.

Simply stated, the purpose of a computer service organization is to provide maintenance services to customers with computer equipment. The primary objective of the organization is to minimize the down-time of that customer equipment and thereby minimize customer inconvenience. At the same time the service organization seeks to operate as cost effectively as possible which, of course, minimizes the service cost to the customer. To provide the maintenance services required, computer service organizations will generally establish a branch office to supply the needs of customers within a certain geographical region. In order to effectively structure that office three fundamental questions need to be answered:

- 1.) What types of service engineers should the office have (i.e., should the engineers be generalists, specialists, or some combination);
- 2.) How large should the branch office be (i.e., how many service engineers are required); and
- 3.) How should service requests or calls of different types be scheduled and which engineer types should be assigned (e.g., first-come-first-served, shortest-expected-service-time; generalist, specialist).

For our company the task of addressing these questions was given by our upper level management to our internal management science consulting group. In the discussions that follow in this paper, I will indicate our findings specifically with regard to question 1. Although some of the numerical results of our studies must remain proprietary, I will indicate the overall conclusions reached and note the pilot test plans that have resulted from our work. I will comment briefly about questions 2 and 3 throughout my discourse on question 1. The complete answers to those questions are still

being determined; perhaps, in the future, they can be incorporated into papers similar to this one.

## SUMMARY RESULTS

At the outset it was generally felt amongst the members of the analysis team that the proper use of engineer specialization could generate significant branch savings over an all engineer generalist environment. To define terms, an engineer specialist is an individual who can perform *certain* repair tasks in less time than the average engineer generalist who is able to perform *any* repair task. Our findings confirmed that, indeed, engineer specialization can result in considerable savings to a branch office. These savings may be expressed as lower response time or increased call handling (or call rate) capability, with no increase in personnel nor in cost. If the proper conditions are present at the branch, specialization can even possibly result in reduced personnel and lower costs.

Figure 1 shows two plots of response time versus call rate. The one to the left is for an all generalist office (no specialization) while the one to the right is for an office with some specialization. Note that the response time/call rate curve for the specialist office lies below and to the right of the non-specialist office.

The implications of this shift are summarized in Table I and discussed below.

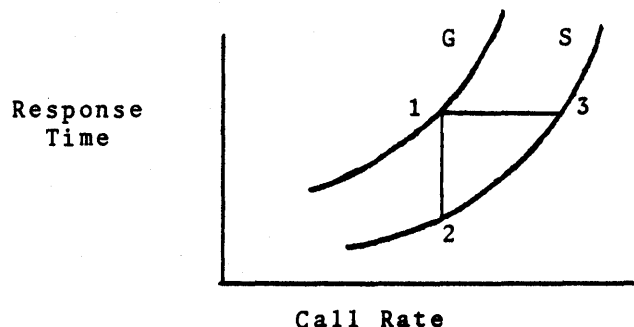


Figure 1

Table I

	From Point 1 to Point 2	From Point 1 to Point 3
Response Time	Decrease	Constant
Mean-Time-To-Repair	Decrease	Decrease
Call Rate Capacity	Constant	Increase
Engineer Idle Time	Increase	Decrease
Engineer Salary	Decrease	Decrease
Engineer Training	Decrease	Decrease

Switching from an all generalist office at Point 1 on the generalist curve to an office with specialists at Point 2 on the specialist curve reduces response time without affecting the call arrival rate and without increasing the total number of engineers. Since specialists are used, overall mean-time-to-repair (MTTR) decreases and hence idle time increases. This last point could be considered as a loss to the office in that there are engineers available to take calls but there are no calls to take. However, this idle time increase over offices without specialization may be minimal and therefore insignificant when compared to the other benefits of specialized offices. Note also that engineer salaries are generally lower for specialist engineers and training requirements are also reduced.

Moving from an all generalist office at Point 1 on the generalist curve to an office with specialists at Point 3 on the specialist curve increases call rate capacity without increasing response time and without increasing the number of engineers in the branch office. Again, overall office MTTR decreases but since more calls are being taken engineer idle time decreases. Also, as was said for Point 2, total salary and training requirements are reduced over the all generalist office with its operating characteristics described by Point 1.

METHODOLOGY

The results of the study summarized above were determined from a computer simulation model written in GPSS (General Purpose Simulation System) to represent a typical branch environment. As stated earlier, minimizing customer system downtime becomes the goal of each and every branch office. In the simulation model system, downtime is divided into its three component parts: (1) waiting time, (2) travel time, and (3) repair time. Waiting time is that period of time

		Engineer Group		
(MTTR in Hours)		1	2	3
Call Type	1	2	3	4
	2	3	2	3
	3	4	4	2

Figure 2

(Engineer Priorities)

		Engineer Group		
		1	2	3
Call Type	1	1	2	3
	2	3	1	2
	3	2	3	1

Figure 3

from when the service request is received by the service organization until an engineer is dispatched to the call. Travel time is, of course, the time it takes the engineer assigned to reach the customer site. Taken together waiting time plus travel time is referred to as response time. Repair time is the time the engineer requires to correct the malfunction once on site. Waiting time is a function of engineer availability (which is, of course, dependent on many factors). Travel time is dependent on the geographical distribution of customers, and repair time is a function of component technology and engineer skill levels. The simulation model considers travel times and repair times as distributed system inputs and evaluates the variation in waiting time (also distributed) as a function of all system input variables.

The easiest way to describe the nature of the other system inputs is to consider Figures 2 and 3 below (Note: the repair time and priority rankings used in the Figures are strictly arbitrary).

Figure 2 depicts how MTTRs for the various engineer groups can be specified to the model for varying types of arriving calls. This allows the user to create specialist groups if desired and to differentiate repair times for different types of calls (e.g., corrective maintenance, preventive maintenance, and installation calls).

Figure 3 details how the priorities with which engineers are assigned to certain types of calls are inputted into the simulation model. This allows the user to determine the impact on the office operating characteristics of varying the engineer/call type priority assignments.

It is important to note at this point that fundamental textbook queuing theory analysis will not readily permit the use of such an extensive collection of input parameters. Basic queuing theory will consider only the total number of engineers, an average call rate, and an average service time as input parameters. The ability to specify engineer types, to use distributed call rate and service times, and to prioritize engineer assignments by call type is not available in the standard queuing equations.

Table II

	Generalist	Specialist		
		True	Limited	Senior
Specialty MTTR	N/A	2	2	2
Non-Specialty MTTR	5	∞	7	5

From the GPSS simulation model it was possible to obtain the following output data:

1. Average waiting times and waiting time distributions by call type,
2. Average repair times and repair time distributions by engineer group,
3. Idle time by engineer group, and
4. The number of calls of each type taken by each engineer group.

Basic textbook queuing theory would only have allowed the determination of a composite waiting time and a composite engineer idle time.

In the process of performing the study three different types of specialists were considered—true, limited, and senior specialists. Typical MTTRs in hours (arbitrarily chosen here) for a generalist and for each of these specialist types are noted in Table II.

As noted a true specialist could repair his/her speciality more quickly than a generalist, but the true specialist could only repair a very small subset of devices. A true specialist would be paid considerably less than a generalist and would receive less training as well. A limited specialist could repair his/her speciality devices more rapidly than a generalist, but the MTTR on the non-specialty devices would be greater. Salary and training levels would be between the true specialist and the generalist. The senior specialist essentially resembles a generalist in repair times except that his/her repair time on specialty devices would be lower. The salary of a senior specialist would exceed that of a generalist, but training requirements would be equivalent.

**ANALYSIS**

When investigating the impact of specialization on offices of a given size (i.e., offices with a fixed number of engineers), a number of interesting results with respect to response time and engineer idle time were noted when the number and type of specialists in the office was varied while the percentage of specialist type calls remained the same. For example,

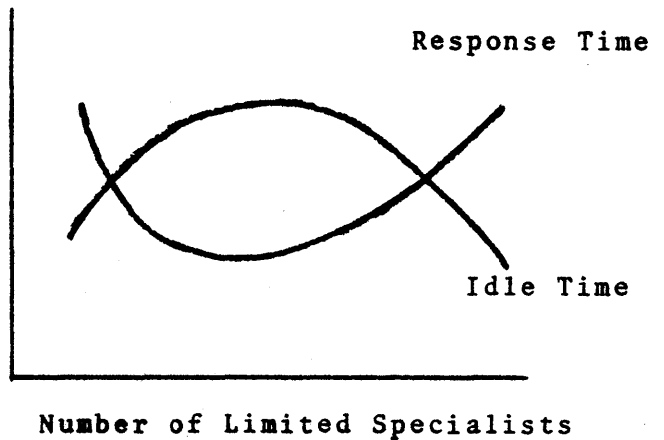


Figure 5

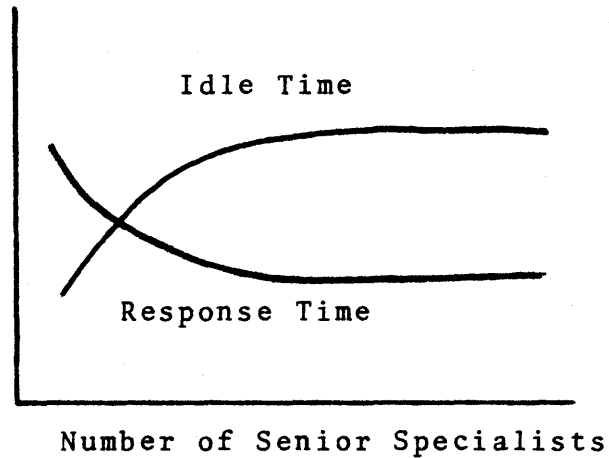


Figure 6

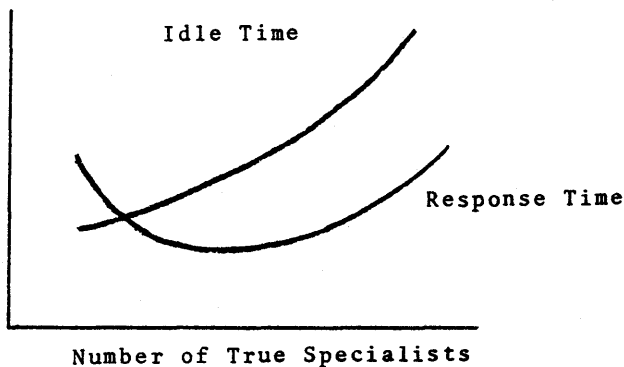


Figure 4

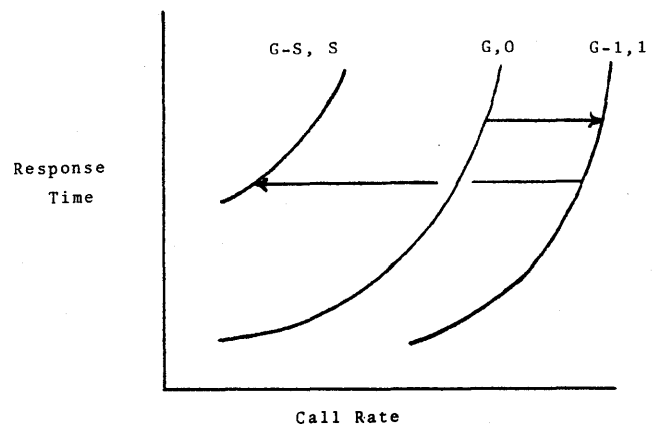


Figure 7

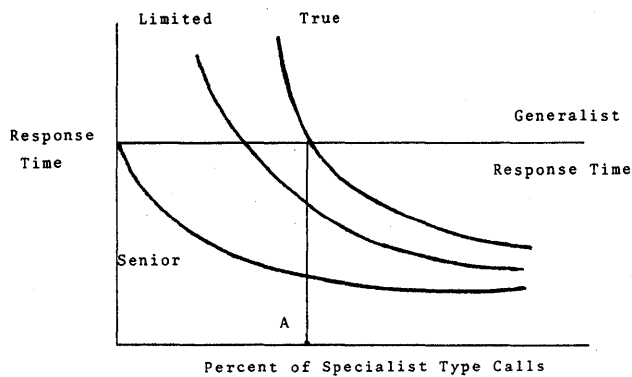


Figure 8

Figure 4 expresses the general response time and idle time phenomena for an office with an increasing number of true specialists. As the percentage of specialists in the office approaches the percentage of specialist type calls, response time decreases since all the specialists are busy. However, when the percentage of specialists exceeds the percentage of specialist type calls, some specialists must remain idle since they cannot take calls of other types. Hence, the office actually has fewer effective engineers, and both the response time and engineer idle time increase.

Figure 5 is a similar representation for limited specialists. Note that the response time curve is similar to that of true specialists. This is true because, when there are more specialists than specialist calls for them to handle, they accept non-specialist calls which require an MTTR greater than that for a generalist. Idle time consequently decreases under these circumstances.

Finally, Figure 6 represents the response time and idle time characteristics for an office utilizing varying numbers of senior specialists. As was true for both the true and limited specialist cases, response time initially decreases and idle time increases as the percentage of specialists approaches the percentage of specialist type calls. However, after that equality point, response time and idle time remain the same since there is no degradation of MTTR below the all generalist MTTR level when a senior specialist is working on non-specialty type calls.

The impact of these idle time and response time phenomena on the response time/call rate characteristic operating curve for a branch office is significant. This can be readily seen in Figure 7 which can be considered representative of a branch with a given total number of engineers, a variable number of true or limited specialists, and a fixed percentage of specialist type calls.

The operating curve for an all generalist office (G generalists and O specialists) initially moves to the right as the number of generalists is decreased by one (G-1) and the number of true or limited specialists is increased by one (1). That movement to the right will continue as long as the percentage of specialists in the office is less than the percentage of specialist type calls. Once it becomes greater the characteristic

curve will move to the left approaching and eventually passing the all generalist curve (note the G-S,S curve to the left of the all generalist curve in Figure 7). It should be noted here that for the senior specialist case the characteristic curve will continue to move to the right regardless of the number of generalists replaced by senior specialists since any senior specialist MTTR is never higher than that of a generalist.

At this juncture it would appear that several rules-of-thumb can be stated regarding the proper use of the various specialist types so that the branch office can operate more effectively.

#### Rule 1. True specialist

In an office with true specialists, the percentage of specialists *must not* exceed the percentage of specialist type calls.

#### Rule 2. Limited specialists

In an office with limited specialists, the percentage of specialists *should not* exceed the percentage of specialist type calls.

For an office with senior specialists no rule can be stated explicitly because, although the operating curve will always be superior to the all generalist office, senior specialists are more costly in terms of salary and training. Hence, some type of performance versus cost tradeoff must be made in this case.

In the preceding analyses the percentage of specialist type calls has been held constant while the percentage of specialists has been varying. It is interesting to note the impact on response time of varying the percentage of specialist type calls while holding the percentage of specialists fixed. Figure 8 is a graphical representation of these phenomena for the three different types of specialists.

It clearly shows that the lower the percentage of specialist type calls the more sophisticated the specialist type must be

Table III

Device	% Reduction
A	53
B	55
C	27
D	22
E	50
F	40

in order to keep response time within reasonable bounds. Point A represents the equality of the percentage of specialist type calls and the percentage of specialists in the office. Clearly, when the latter exceeds the former (i.e., to the left of Point A) the use of true specialists becomes inferior to the all generalist case. To the right of point A, however, the three curves converge since all specialists are kept busy on their specialty.

## IMPLEMENTATION

Before committing to the implementation of an engineer specialization program it was first necessary to determine the amounts by which specialization could actually reduce MTTR. Although I am not at liberty to discuss the complete nature and extent of the tests that were conducted to make these determinations, I have noted in Table III some sample percentage reductions in repair times for specialists over generalists for six individual (but unnamed) devices.

With this and other supporting data at hand our company has decided to implement a field service (i.e., hardware) specialization program. Our management science consulting organization is presently helping individual branch offices determine when and if and what kinds of specialization should be utilized. By modifying the simulation model described in this paper to reflect individual branch office operating parameters various alternative specialization schemes can be evaluated for a given office. In essence, a decision support system is being utilized jointly by our consulting group and by our branch management personnel to determine the office structures that will both maximize customer satisfaction and be cost effective. Given our belief that customer service issues will be the driving force in our industry in the 1980s, our dedication to the task at hand must be complete. Whereas, our emphasis is on the field service organization today, it will most certainly be on the software services business tomorrow. Modification of the simulation model (or the decision support system, if you will) to that end is a certainty.





# On development tools for small systems: the challenge of economically automating a filing cabinet

by DAVID D. RABER

*MegaWest Systems, Incorporated*  
Salt Lake City, Utah

## THE SMALL SYSTEM CHALLENGE

One measure of how effective a DP shop is, or how efficient a set of development tools are, is the minimum size project that can effectively be handled. Granted, this is not the only measure of effectiveness, nor is it necessarily the best. On the other hand, a data processing department which can effectively produce small business systems may find a rich potential for service and success within the corporate structure. Realizing this potential depends, to a large degree, on the amount of specialized small system expertise in the areas of development methodology, hardware capability, and development tools.

Perhaps some constructed examples will provide a useful introduction to the challenges which are related to small system development:

The employee's cafeteria wants to automate their commissary inventory. They looked at a manufacturing parts inventory package and rejected it as being entirely too complicated.

The graphics studio wants a system to keep track of form layout requests. They briefly considered a shop floor management package with bill of material and human resource allocation features, but concluded it was too complex for their three person operation.

The Systems Development Department wants a system to keep track of maintenance requests from user departments. Someone "between projects" has already spent five weeks on this one.

The savings and loan down the street has a turnkey loan application system. They want to add a little program to keep track of the "pots and pans" gift premium inventory.

A recent front page article in the Wall Street Journal was headlined "Executives Discover Computers Can Help Them in Daily Routines" [6]. The article describes an increasing acceptance among business professionals of automating routine office functions. Examples given in the article include text processing, appointment scheduling, and memo distribution. Small functions which are also being automated include electronic doodle pads and tickler files. While a complete automated office of the future is definitely in the realm

of large system development, it reflects an unmistakable trend toward automating simple business tasks.

## SMALL SYSTEM DEVELOPMENT METHODOLOGY

A successful small project strategy must begin with appropriate development methodology and project management. While an eleven phase development methodology complete with project schedules, technical reviews, and sign-offs may be appropriate and necessary for traditional projects, these procedures would smother all but the heartiest of small projects.

One method of handling small projects efficiently would be the creation of a special small systems section, perhaps consisting of only one or two analysts. The manager of this section would have authority to review, approve, schedule, and implement projects which were consistent with the firm's DP goals and with the small project section's special mission. A project would be disqualified if it required too many development or production resources, if it was logically part of a larger system which was at least in the initial survey phase, or if it required a complex interface to other systems. If the project qualified as being small enough, one of the "small project" analysts (or the manager himself) would implement and document it.

Small system expertise can be useful to both the user departments and the DP department itself. It can provide an excellent method for the DP department to "meet" other departments on a functional level. While the DP department may be ready to automate the entire company, the entire company may not be ready to be automated. If there is some reluctance on the part of users to automate, then start small. This will give the DP department exposure to the users as well as giving the users some hands-on experience with what computers can do and with the role they need to play in specifying and implementing systems.

DP departments have a great internal need for small system expertise as well. Some examples of small systems useful in the system development process are data dictionaries, logical and physical file documentation, development task scheduling, hardware or software evaluations, system module indexing, and test file creation.

To summarize, small systems must be as flexible as they are simple. Remember, small systems are frequently going to be designed for the naive-to-unsophisticated user. In addition, the low overhead environment in which small system development must operate does not permit an in-depth analysis by the DP department. The successful small systems specialist will give an application his "best guess" the first time through. If he misses, the analyst and the users will be able to take advantage of the flexibility built into the system and make whatever modifications are necessary with minimal effort.

### HARDWARE FOR SMALL SYSTEMS

In an environment where the availability of efficient development tools is critical, hardware selection criteria can hardly be confined to system architecture and memory cycle time. The development tools under discussion here are no exception to the rule that hardware is the major factor in determining what is efficient, effective, and practical in the software realm. While the particular tools discussed here are designed for Microdata's Reality line of minicomputers, it is not the intent of this paper to concentrate on the virtues or vices of any particular hardware. Rather, several features of the hardware and operating system which significantly impact the software development process will be highlighted. Attention will be given to the portability of these concepts to other hardware environments.

A particularly useful feature provided by Microdata's Reality operating system is dynamic arrays. Dynamic arrays allow variable length records, each record with variable length fields, each field with variable dimensions, each dimension with a variable number of subvalues. Dynamic arrays eliminate the constraints of fixed columns and field widths.

In Reality (no pun intended this time) each file is comprised of any number of "items" which may be thought of as records. Each item in the file is a dynamic array. Within each dynamic array there are any number of "attributes" which can be thought of as data fields. The attributes within a dynamic array are delimited by a special character referred to as an attribute mark. Similarly, each attribute on the record can have any number of "values," which can be thought of as values within a one dimension array. Values within the attribute are delimited by a special character referred to as a value mark. These values can be broken down into subvalues, and subvalues can be broken into any number of subfields. Rather than defining data as occupying predefined columns on a record, the data is defined as occupying a relative position on the record as determined by the special delimiter characters.

This feature allows designers to free themselves of considerations such as length of fields, maximum number of occurrences per variable, or the number of columns per record. This method is somewhat of a compromise between elemental data storage and a full data base management system. While providing some genuine relief from the burden

of elemental data storage, the processing required is straightforward enough to implement in microcode [4].

The second feature is an optional dictionary section available for each file in the system. This feature is closely related to the first in that dictionaries provide an opportunity to name fields of a dynamic data array based on the field's relative position. The dictionary can also be used to specify format and data conversions to be performed on the data as it is input or displayed. Part of the development tools discussed in the next section provides a simple, automated method of creating dictionary entries. As shall be shown, dictionaries play a vital role in data input and retrieval functions.

While Reality makes specific provisions for data dictionaries, this concept could be (and is) implemented on other systems. Rather than defining data as being located in a given attribute, as in Reality, data could be defined as occupying a range of columns. The conversions, format, and print lengths could easily be stored alongside the location reference.

The third feature is a very powerful report generator called English [2]. English makes extensive use of the data dictionary described above. English can be used to list or sort files or subsets of the file based on optional selection criteria. English also features control-break functions with various totaling and subtotaling options.

English is completely dictionary driven in that all parameters required to retrieve and display data are stored in a system-provided dictionary and are accessed automatically by English. Users of English only concern themselves with the names of variables they want displayed, sorted, or totaled.

The following examples are typical English statements:

```
LIST ACCOUNT DESCRIPTION BALANCE
LAST,POST
SORT ACCOUNT BY TYPE BREAK-ON TYPE TOTAL
BALANCE HEADING "ACCOUNT SUMMARY AS
OF 'D'"
COUNT INVENTORY WITH PRICE = '.30'
SORT-LABEL CUST BY ZIP NAME ADDRESS
```

In the first example the account file's dictionary is presumed to contain entries which define the location, conversion, justification, and print length of the variables named description, balance, and last.post.

The real power of English lies in the conversion specifications stored in the data dictionaries. One of the more complex conversions allows the use of other files as translate tables. Another powerful conversion supports a full set of algebraic and logical operations which have at their disposal any data element in the entire system. English can, for example, produce bar charts by a fairly simple one-line conversion stored in the data dictionary. In addition to complex conversions, simple editing of date and decimal fields are also provided for. Any number of conversions may be specified for each item in a dictionary.

The value of a programming tool such as English should not be based solely on how it performs as an ad hoc report generator. Although an ad hoc report generator is necessary for the success of small systems, it is not sufficient. Some convenient method must also be provided to specify, design, and program standard production reports. As shall be discussed shortly, English plays a major role in productivity on Microdata systems by providing an efficient method of displaying data on both an ad hoc and on a regular basis.

The fourth feature is direct file access. It should be obvious that small systems which are intended to replace manual filing systems cannot be successfully implemented with only sequential access. One drawback to Reality's file structure is that multiple keys or indexed sequential access is not provided for directly by the operating system.

The fifth feature is a virtual memory management system. In Reality all external on-line storage is logically in executable memory. This has several implications. First, "loading" programs and "opening" files is transparent since all programs and files are logically in memory at all times. Second, the length of program and data fields is not constrained by storage partitions. Obviously several 32K programs running concurrently on the smallest Reality configuration (16K) are going to experience considerable frame faulting. While there is substantial memory management overhead, this scheme does achieve a worthy goal of insulating users from memory management.

The sixth feature is a powerful proc language referred to simply as Proc [3]. Any command or character string which can be entered at the terminal can be stored in a proc and called up for submission as if it were terminal input. In addition, Proc has sufficient logical, algebraic, IO and branching capabilities so that many programming functions can be done directly in Proc. Since procs are interpreted at execution time, their use is limited in practice to pre or post processing routines for standard system utilities or application programs.

One simple example of a Proc application is as a pre-processor for English. To reduce the number of keystrokes required to produce an English report, it is often desirable to store long English commands in a file. A proc can be written to examine an English statement for references to the pre-stored command file. If a reference is found, the pre-stored text is merged with the rest of the terminal input. As this example demonstrates, a powerful proc processor can greatly enhance the friendliness of a system by eliminating unnecessary terminal input.

Notice that most of these hardware and operating system features are valuable to the production of small systems due to their ability to incorporate flexibility into the system design. If a field needs to be longer or more variables are needed, they can be added without affecting existing data or programs. If new values need to be computed or translated, it can be done as the variables are called up for display through the use of dictionary-specified conversions. If report formats need to be changed or special sort criteria need to be added, simple changes to report generating procs can be made.

## SOFTWARE TOOLS FOR SMALL SYSTEMS

### *The filing cabinet system*

While hardware often comes with useful development tools such as Microdata's English and Proc, these tools frequently provide only the foundation of what is needed for small system development. In the case of Reality, a simple dictionary driven data input process was lacking. A rather satisfactory solution has been based on a set of file maintenance programs originally designed by Mr. Paul Hyer while he was the director at Brigham Young University Hawaii Campus Computer Center [1].

Rather than composing a pretentious name for the tools, they will be referred to as simply the "filing cabinet" system. As the name implies, filing cabinet systems consist of an automated but simple method of filing and retrieving information. Each item (record) in the automated system represents a filing card from the manual system. Each attribute (data field) represents a field on a preprinted form. Each value (dimension) can be thought of as multiple occurrences of a field. In the case of both the manual and the automated system, records are arranged according to a unique but relevant key.

There are two major processes in a filing cabinet system, data input and data retrieval. Data input includes creating new records, updating existing records, and deleting records

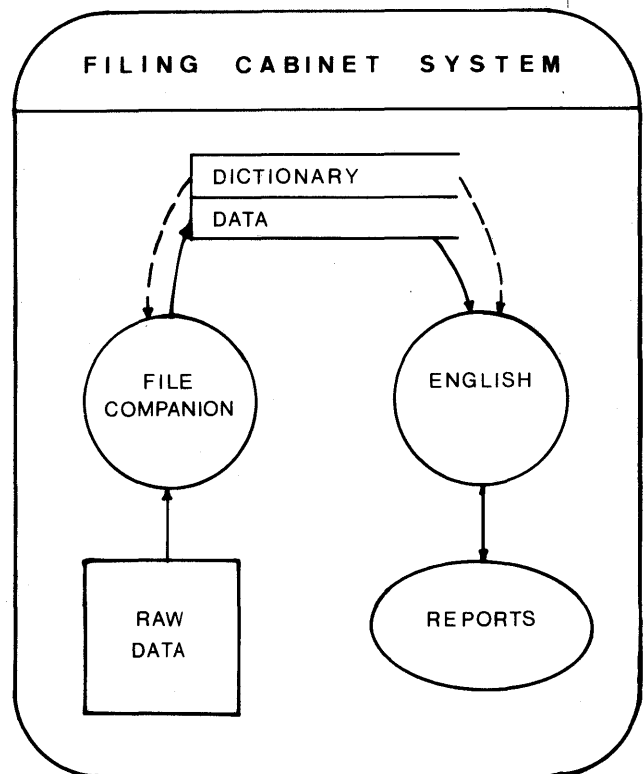


Figure 1

no longer needed. Adding or updating may include editing for valid data and converting values into an internal storage format. Data retrieval includes displaying data in external format as well as performing sorting and selection task.

Two ancillary tasks which may be performed in filing cabinet systems are maintaining inverted list and transforming data into new attributes. Maintaining inverted list is sometimes necessary and desirable if the file is large and access is frequently required through secondary keys. Reality's English processor does provide a facility for maintaining and using inverted list. As indicated previously, transformations required on the attributes are incorporated in the data retrieval or data input processes whenever possible so as to take advantage of the flexibility English provides.

The major functions of a filing cabinet system are represented in Figure 1. While English should be a familiar concept by now, File Companion has not previously been mentioned

since it is an in-house development and thus not part of the Reality package. File Companion is a generalized data input program which, like English, makes use of the dictionary section of data files. Since all parameters necessary for the operation of File Companion are stored in the dictionary of each data file, this program provides a readily available data input capability for any Reality file that has a current dictionary.

#### Overview of file companion

File Companion (FC for short) operates on a concept basic to many other editors. The fundamental difference is that FC functions one level below record oriented editors. The user of a record oriented editor operates on records within files. FC takes advantage of Reality's file structure by editing

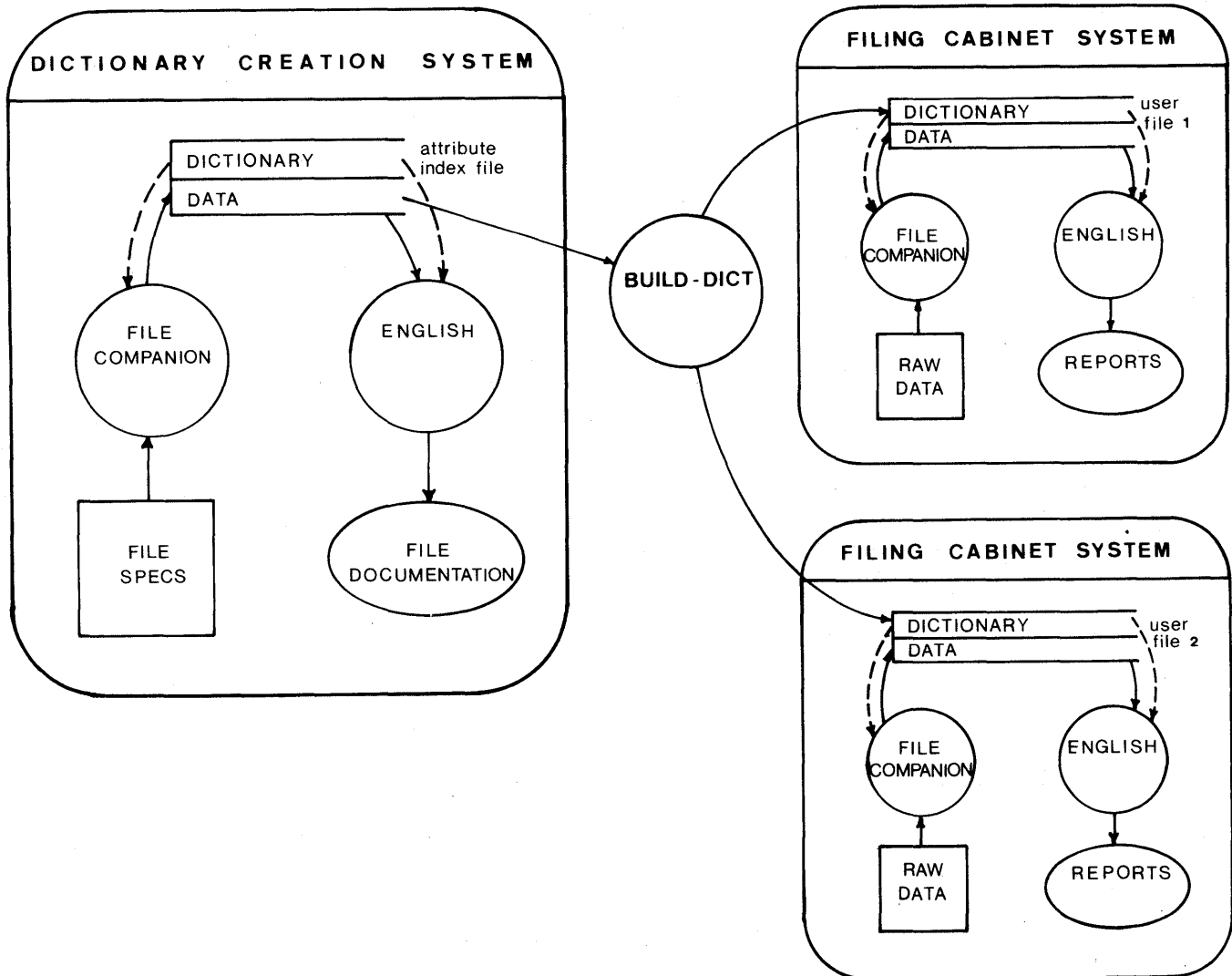


Figure 2—Attribute definitions are stored in the attribute index file (represented on the left). BUILD-DICT is used to build dictionaries for all user files in the system (represented on the right).

```

:fc at
NUMBER OF ATTRIBUTES IS 15

ENTER ITEM-ID:1
---NEW ADDITION---

 1 FILE.NAME           :form
 2 ATTRIBUTE           :0
 3 NAME(S)             :form.id
 4 TYPE                :k
 5 DESCRIPTION         :internally assigned form number
 6 FORMAT.CLASS        :
 7 CONVERSION          :
 8 CORRELATIVE         :
 9 JUSTIFICATION       :1
10 PRINT.LENGTH        :6
11 ENGL.HEADING        :
12 ENGL.DFLT.SEQ       :
13 INPUT.LENGTH        :6
14 INPUT.CONVERSION    :
15 PATTERN             :2n1a3n

ENTER ITEM-ID:2
---NEW ADDITION---

 1 FILE.NAME           :form
 2 ATTRIBUTE           :1
 3 NAME(S)             :on.hand
 4 TYPE                :r
 5 DESCRIPTION         :quantity on hand
 6 FORMAT.CLASS        :
 7 CONVERSION          :
 8 CORRELATIVE         :
 9 JUSTIFICATION       :r
10 PRINT.LENGTH        :5
11 ENGL.HEADING        :
12 ENGL.DFLT.SEQ       :1
13 INPUT.LENGTH        :
14 INPUT.CONVERSION    :
15 PATTERN             :num

```

Figure 3—Example of File Companion being used on an attribute index file to create new dictionary entries. Refer to left-hand side of Figure 2.

individual attributes (data fields) within an item (record), each item being a subdivision of a file. One obvious advantage to this editor is that it prompts with descriptive attribute names obtained from the dictionary rather than simple line numbers as in the case of many other editors. Another ad-

vantage is that FC screens input data based on editing parameters stored in the system provided data dictionary.

Implementing FC as a line-at-a-time editor rather than as a full screen editor was made for three reasons. First, there are several medium speed hardcopy terminals in our systems

```

FILE.DOC.3
001 SORT AT BY FILE.NAME BY ATTRIBUTE BY TYPE BREAK-ON T.FILE "'PB'*"
002 LISTING ATTRIBUTE NAME DESCRIPTION T.TYPE CONVERSION PRINT.LENGTH T.JUST PA
    TTERN
003 ID-SUPP DBL-SPC
004 HEADING " FILE SPECIFICATION FOR: 'B' FILE PAGE 'PLL'"

```

Figure 4—English statement used to produce file documentation as displayed in Figure 5.

FILE SPECIFICATION FOR: FORM <FORM INVENTORY> FILE

PAGE 1

AT#	NAME(S)	DESCRIPTION	TYPE	CONV	LEN	JSTF	PATTERN
	FORM.ID	INTERNALLY ASSIGNED FORM NUMBER	KEY		6	LEFT	2N1A3N
1	ON.HAND	QUANTITY ON HAND	REQUIRED		5	RIGHT	NUM
1	LOW	LOW=1 IF ON.HAND < 10% OF QNT.REORDER	SYNONYM	A;N(ON.H AND)*"10 "<N(QNT.LOT)	1	RIGHT	
2	REV.DATE	DATE OF LAST REVISION	REQUIRED	D2	9	RIGHT	
3	QNT.LOT	USUAL REORDER QUANTITY	OPTIONAL		5	RIGHT	NUM
4	RE.DATES	REORDER DATES (VALUE MARK BETWEEN DATES)	OPTIONAL	D2	9	RIGHT	
5	RE.QUANT	REORDER QUANTITIES	OPTIONAL		5	RIGHT	
6	PROCESS	PROCESS USED TO PRODUCE FORMS O=OFFSET NCR=NCR MULTIPLE PART FORM X=XEROX E=ENGRAVED	REQUIRED		5	LEFT	
7	LOC	BULK INVENTORY LOCATION R=BASEMENT 8=8TH FLOOR STORE ROOM D=DEPARTMENT'S STOREROOM	OPTIONAL		3	LEFT	
8	DEPT	DEPARTMENT THAT INITIATED THE FORM REQUEST	REQUIRED		4	LEFT	
9	DESCRIPTION	DESCRIPTION OF FORM	REQUIRED		15	TEXT	
10	SIZE	SIZE OF FORM	REQUIRED		6	LEFT	

\* Figure 5—File documentation produced by English from the attribute index file. Refer to left-hand side of Figure 2.

which do not support cursor positioning. Second, there are several different video terminals in use, each using different cursor control protocol. Third, using one or more lines for each attribute enables FC to handle attributes of any size as well as an unlimited number of attributes for each item.

FC is invoked by typing "FC" followed by a space and the name of a file to be operated on. FC performs the following task:

**Initialization**

A) Open the dictionary section of the file specified. Select all attribute definitions which are not identified as synonyms. This eliminates from the input process data which is constructed rather than physically on file.

B) For each item selected, obtain from the dictionary a prompt (i.e. the attribute name), a pattern match, and a conversion specification.

**Data entry**

A) An item-id is prompted for. If an existing id is entered (i.e. the id of an item already on file), the entire item is

```

:fc form
NUMBER OF ATTRIBUTES IS 10
ENTER ITEM-ID:22f834
---NEW ADDITION---
1 ON.HAND :1200
2 REV.DATE :20apr79
3 QNT.LOT :5000
4 RE.DATES :22apr79
5 RE.QUANT :5000
6 PROCESS :ncr
7 LOC :d
8 DEPT :fin
9 DESCRIPTION :cash receipt
10 SIZE :2x6
    
```

Figure 6—Example of File Companion being used on a production file. Refer to right-hand side of Figure 2.

```

FORM. INV
001 SORT FORM
002 LISTING ON.HAND REV.DATE QNT.LOT RE.DATES RE.QUANT PROCESS LOC DEPT
003 DESCRIPTION SIZE LOW
004 DBL-SPC
005 HEADING "FORMS INVENTORY REPORT" 'D' PAGE 'PLL'"

```

```

FORM. LOW
001 SORT FORM BY DEPT WITH LOW = "1" LISTING DEPT DESCRIPTION PROCESS
002 ON.HAND QNT.LOT
003 HEADING "10% CRITICAL INVENTORY LIST" 'D' PAGE 'PLL'"

```

Figure 7—Examples of English statements used for production reports displayed in Figure 8.

displayed in compact format. If a new id is entered a message is displayed indicating it is a new item. In addition, new ids are compared against the pattern match specified for a key, if any.

B) The user is prompted, attribute at a time, for input data. The prompt consist of the attribute number, the attribute name, and the first 20 characters of existing data (if any). The user has several options. First, the existing data can be replaced by entering new data. Second, the attribute can be skipped by entering a null (i.e. just a carriage return). Third, special functions such as branching directly to another attribute, replacing characters in the current attribute, displaying the entire attribute, or exiting can be performed by

entering special control characters. Complete details are given in [Ref. 5].

C) After the attribute list is exhausted another item-id is prompted for. Either another item-id can be entered, or a special command can be entered to perform task such as selecting a new file or selecting a subset of attributes to operate on.

#### *Creating new applications*

As should be evident from the dictionary driven nature of both English and File Companion, designing and installing

```

FORMS INVENTORY REPORT      05 JAN 1980      PAGE 1

FORM.. ON.HAND REV.DATE. QNT.LOT RE.DATES. RE.QUANT PROCESS LOC DEPT DESCRIPTION.... SIZE.. LOW
22FA31   300 01 JUN 77   5000 06 JUN 77   5000 0      D  FIN  TRAVEL ADVANCE  8.5X11  1
          VOUCHER
          15 AUG 78   5000
          01 MAR 79   5000
22FA32   750 07 JUL 78   700 15 JUL 78   700 0      R  FIN  SPECIAL TRAVEL  8.5X11  0
          REQUISITION
          700
22FA33   900 01 APR 76   10000 07 APR 76   5000 0      D  FIN  JOURNAL VOUCHER  8.5X11  1
          21 JUL 76   10000
          01 AUG 77   10000
          07 JUN 78   10000
22 834   1200 20 79   5000 22 79   5000          2 6  0
37P100   740 17 JUL 78   2000 17 JUL 78   2000 0      R  PER  EMPLOYMENT      8.5X11  0
          APPLICATION
37P101   100 22 JUL 78   1200 25 JUL 78   1200 NCR    R  PER  TERMINATION     2X6    1
          NOTICE

10% CRITICAL INVENTORY LIST      05 JAN 1980      PAGE 1

FORM.. DEPT DESCRIPTION.... PROCESS ON.HAND QNT.LOT
22FA31 FIN  TRAVEL ADVANCE 0      300 5000
          VOUCHER
22FA33 FIN  JOURNAL VOUCHER 0      900 10000
37P101 PER  TERMINATION  NCR    100 1200
          NOTICE

```

Figure 8—Examples of production reports produced by English. Refer to right-hand side of Figure 2.



filing cabinet systems can be reduced to the task of constructing an appropriate dictionary. Once a dictionary is constructed, FC can be used for data input and English can be used for data retrieval. The critical task is clearly dictionary construction, which is itself one of the most straightforward applications of the filing cabinet system.

As illustrated in Figure 2, the filing cabinet system that creates dictionaries is functionally identical to user filing cabinet systems. User file dictionaries are created from a master attribute index file by the program BUILD-DICT. Although conceptually the task performed by BUILD-DICT would be a simple application of English, Microdata users are still awaiting enhancements which will allow English to produce files rather than just reports and displays.

The following steps can be used to create and document new applications:

#### File specification

For each data element in the file, assign an appropriate description, print length, justification, and variable type. Specify an input pattern match and conversion if they are necessary. These terms are further defined in the attribute index file documentation [5 and Appendix A].

#### File design

Use FC to record file specifications on the attribute index file. FC will not allow the file, attribute number, or attribute name to be entered as null. Default values for other fields are given in the attribute index file documentation.

#### File implementation

After all on the data elements for a file have been entered, run BUILD-DICT to move specifications from the attribute index file to the user file dictionary. BUILD-DICT is a proc that performs the following subfunctions:

- A) the user is prompted for the name of the data file for which a dictionary is to be built.
- B) if the data file does not already exist, a file is created using default hashing parameters.
- C) the English SELECT verb is used to select all items in the attribute file which defines attributes for the dictionary to be built.
- D) attribute definitions are copied from the attribute index file to the dictionary of the data file.

#### File documentation

Run FILE-DOC to produce file documentation based on information in the attribute index file. FILE-DOC is simply a proc that calls English to list attribute specifications for a given file.

Several options are available if the required processing cannot be handled by a simple application of File Companion and English. First, File Companion may optionally call a Basic subroutine as items are retrieved or stored. This provides the flexibility required to handle task such as maintaining inverted list or inserting default values. Second, a

```
:FC FORM
NUMBER OF ATTRIBUTES IS 10

ENTER ITEM-ID: .S

----ATTRIBUTE SUBSET SELECTION----
ENTER ATTRIBUTE NAME OR NUMBER
ATTRIBUTE 1 :ON.HAND
ATTRIBUTE 2 :
NOTE--TYPE ".S" WHEN YOU ARE READY TO CANCEL ATTRIBUTE SUBSET SELECTION

ENTER ITEM-ID:22F832
50~3841~700~3849~700)700~0~B~FIN~SPECIAL TRAVEL REQUISITION~8.5X11

1 ON.HAND          50          : .+700
                   750

ENTER ITEM-ID:22F831
600~3440~5000~3445)3880)4078~5000)5000)5000~0~D~FIN~TRAVEL ADVANCE VOUCHER~8.5X11

1 ON.HAND          600          : .-300
                   300

ENTER ITEM-ID: .X
```

Figure 9—Example of File Companion being used to update a subset of variables on a production file.

Basic program may be written to perform ancillary processing, as in the case of BUILD-DICT mentioned earlier. There are, of course, some complex applications which are beyond the capacity of a generalized data input function such as File Companion. In this case, development of customized screens and data input processing would be justified.

An example of creating and using a filing cabinet system will give a better idea of how these systems are built and used. For lack of a better example we will assume that the graphics studio is so well pleased with the form request tracking system we put together that they want a little inventory system to keep track of the more than 2,000 forms they produce and stock (including the form to request new forms). While variety of the inventory is great, they tell us that turnover is quite low. After spending a few hours with the users, File Companion, and English we might come up with a system such as the one illustrated in Figures 3 through 9. And sample reports.

## APPENDIX A

### FILE SPECIFICATION FOR: AT (ATTRIBUTE INDEX) FILE

AT#	NAME(S) .....	DESCRIPTION .....	TYPE .....	LEN	JSTF.
0	KEY	SEQUENTIAL KEY TO ATTRIBUTE FILE	KEY	3	RIGHT
1	FILE.NAME FILN	FILE NAME	REQUIRED	5	TEXT
2	ATTRIBUTE AT	ATTRIBUTE NUMBER	REQUIRED	2	RIGHT
3	NAME(S) NAME	NAME OF ATTRIBUTE (VALUE MARKS BETWEEN NAMES)	REQUIRED	15	LEFT
4	TYPE	VARIABLE TYPE R=REQUIRED O=OPTIONAL I=INTERNALLY COMPUTED K=KEY	REQUIRED	1	TEXT
5	DESCRIPTION	DESCRIPTION FOR ATTRIBUTE	OPTIONAL	30	TEXT
6	FORMAT.CLASS FRMT	FORMAT CLASS	OPTIONAL	4	TEXT
7	CONVERSION CONV	ENGLISH CONVERSION (CONVERSIONS ARE DONE AFTER SELECTS AND SORTS)	OPTIONAL	6	TEXT
8	CORRELATIVE CORR	ENGLISH CORRELATIVE (CORRELATIVES ARE DONE BEFORE SELECTS AND SORTS) USUALLY, ALL CONVERSIONS EXCEPT DATE AND DECIMAL WILL GO HERE	OPTIONAL	10	TEXT
9	JUSTIFICATION J	PRINT JUSTIFICATION FOR ENGLISH R=RIGHT L=LEFT T=TEXT (LEFT, LONG TEXT TO NEW LINE)	OPTIONAL	1	TEXT
10	PRINT.LENGTH LN	PRINT LENGTH	OPTIONAL	3	RIGHT

## ACKNOWLEDGMENTS

The author wishes to express appreciation to Michael J. Archuleta of MegaWest Systems for the opportunity to explore many of the numerous challenges small system development presents and to Paul Hyer of Management Systems Corporation for his helpful introduction to some of the solutions which are expressed here.

## REFERENCES

1. Hyer, Paul, "File Maintenance Procedures." Unpublished program documentation available from author.
2. Microdata Corporation, *English Reference Manual*. Irvine: 1977.
3. Microdata Corporation, *Proc and Batch Manual*. Irvine: 1977.
4. Microdata Corporation, *Programmer's Reference Manual*. Irvine: 1977.
5. Raber, David D., "Users Guide to File Companion." Unpublished program documentation available from author.
6. Wysocoki, Benard Jr., "Automated Offices: Executives Discover Computers Can Help Them In Daily Routines." *Wall Street Journal*. July 6, 1979.

---

AT#	NAME(S) .....	DESCRIPTION .....	TYPE .....	LEN	JSTF.
11	ENGL.HEADING ENGL	COLUMN HEADING TO BE USED FOR ENGLISH. ITEM NAME IS THE DEFAULT HEADING.	OPTIONAL	10	TEXT
12	ENGL.DFLT.SEQ SEQ	SEQUENCE NUMBER FOR ENGLISH REPORTS WITH NO OUTPUT SPECIFICATION	OPTIONAL	2	RIGHT
13	INPUT.LENGTH ILEN	MAXIMUM LENGTH (USED BY SCREENPRO AND FC)	OPTIONAL	3	RIGHT
14	INPUT.CONV ICONV	INPUT CONVERSION	OPTIONAL	6	TEXT
15	PATTERN	INPUT PATTERN MATCH EXAMPLES: 2N9A=2 NUMERIC, 9 ALPHA 3X1A=3 ALPHANUMERIC, 1 ALPHA 3N'A'=3 NUMERIC, LITERAL 'A'	OPTIONAL	8	TEXT

# A structured information system design for a newspaper organization: a case study

by MOHAN R. TANNIRU

University of Wisconsin-Madison  
Madison, Wisconsin

## INTRODUCTION

Designing an information system, whether it is one of decision making or decision supporting, often starts with the identification of the objectives of the system. Information is then generated and processed to meet these objectives. In many transaction processing systems (also known as life-stream systems or programmed systems) the objectives are expressed in terms of reports, both scheduled and ad hoc (output design). The content portion of this output design is used to determine the information needs. Some of this information may be computed internally (process design) or input directly from an external source (input design).

This seemingly simple structure to the design of information systems, however, becomes more complex as the problem becomes unstructured. The absence of a structure is often due to one's inability to identify, *a priori*, the demands that will be imposed on the system (objectives of the system), or to ascertain the appropriateness of various mechanisms used in achieving these objectives, or both. One's inability to identify the demands on the system makes the input design difficult and the lack of an acceptable procedure for achieving these demands makes the process design dynamic. An example of this is a planning system. Here the objectives are numerous, conflicting and changing constantly, and the methodologies used to achieve those objectives vary significantly in their scope, precision and complexity. This is one of the reasons for the design of support systems for planning. The design of these support systems is often complex and no standard methodology exists, as of today, for the design of these systems.

Significant research has been done in the top-down design of large complex programs [1], and in the automation of system design from a set of user specifications [2,3,4]. The use of top-down approach to program and system design provides a modular approach and, thus, reduces the consistency and maintenance problems. It was shown [5,6] that similar benefits can be derived by relating the top-management's goals/objectives, expressed as functions of the chart of accounts, to various decisions made in the firm using standard financial flows of the firm. The objective of this study is to develop a framework that can be used to accept a planner's requirements in terms of planning and operational activities

along with their interdependencies, and provide a logical specification of his/her requirements. This specification may then be used as input to an automatic program generator for detailed system design or it can be treated as a sub-schema in a data base environment.

The discussion here is organized into four sections. Following this introduction, section one briefly describes the case under consideration (newspaper industry) and its representation as a hierarchical data model. We will also discuss here some of the basic terminology and notation that will be followed in later discussion. Section two develops a methodology for the system design under two activity classification schemes. Section three suggests the approach that will be used to the actual system implementation. Section four illustrates some of the expected benefits of using this approach along with some issues that need further study.

## A HIERARCHICAL DATA REPRESENTATION

The major objective of a newspaper is to disseminate news to various subscribers at a relatively low cost and the extent of coverage that can be provided, to a large extent, depends on the financial strength of the organization. Much of this financial strength is derived from the advertising dollar that it can generate as a major source of its revenues, and this in turn depends on the amount of circulation. Four major decision centers in an organization of this type are: Circulation (subscription, paper distribution, and handling starts/stops); Production; Editing; and Advertising (display and classified advertiser selection and billing).

The financial strength is evaluated by creditors and/or investors in a manner similar to one chosen for any money making organization, i.e., by observing various financial ratios. Since loyalty of subscribers often plays a greater role in this business, the circulation department's functions play a critical part in ensuring accurate billing and reliable distribution. This becomes rather important when there are competing newspapers catering their service to same market. Problems related to this activity were the ones that initiated this study. However, top management's recognition that a comprehensive system has to be designed ultimately for proper integration encouraged us to view the system in its

entirety for the purposes of design. The actual implementation will be done one module at a time with the first module being the one associated with circulation decisions.

The financial state of this organization is represented by a hierarchical structure shown in Figure 1. The nodes of this tree structure correspond to various stock and flow (Balance Sheet and Income Statement) accounts used by the orga-

nization. The state of this system at time  $t(b')$  correspond to the vector of values of each of the states (nodes) in the system at time  $t$ . The state of the system changes from time  $t$  to  $t + 1$  due to system flows (arcs) that correspond to various monetary activities within the firm. To be consistent with the accounting conventions, a flow is said to affect two states at the same time. The convention adopted here is that the

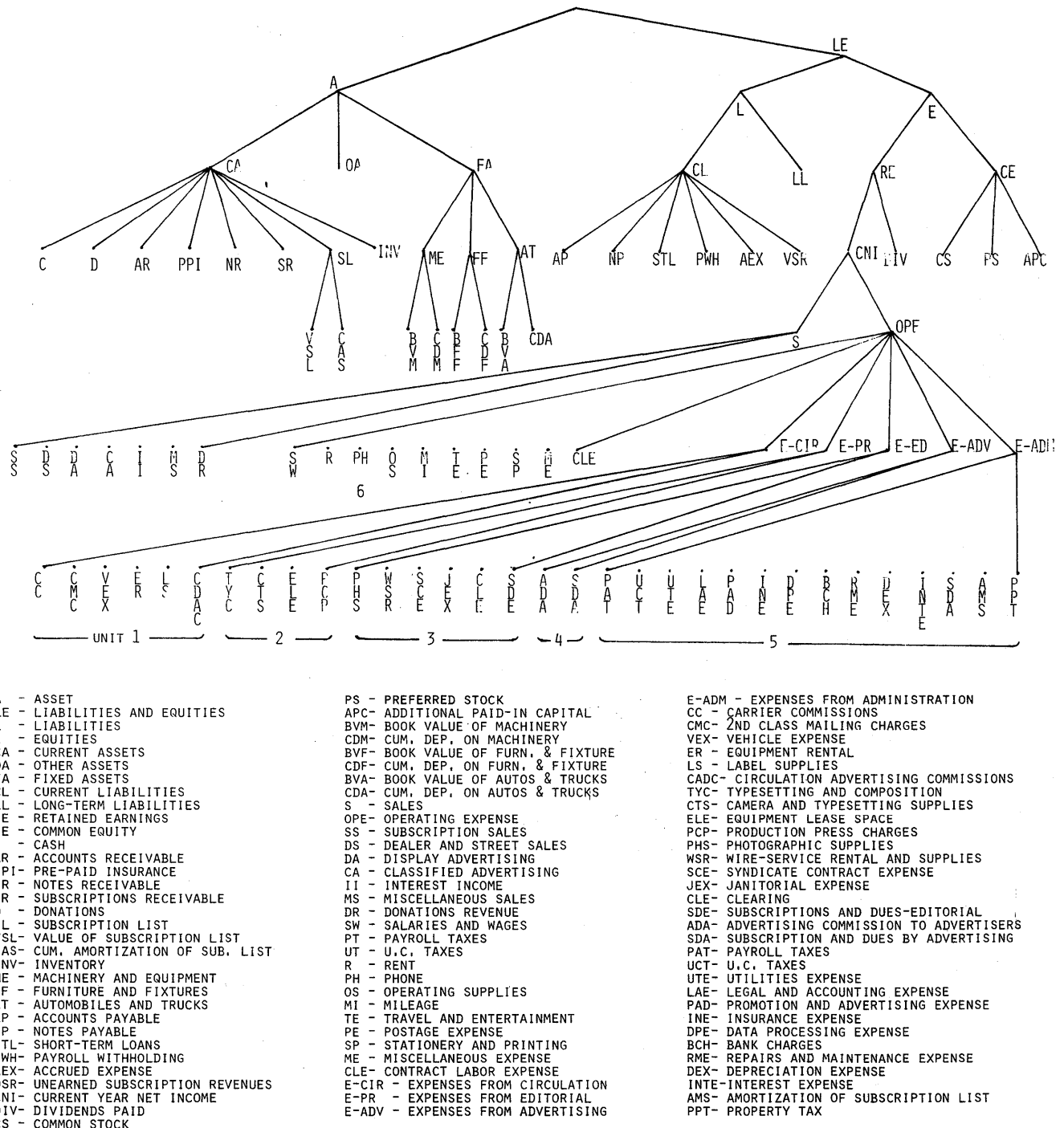


Figure 1—A hierarchical structure representing the financial state.

starting node of the flow was affected negatively by the flow amount (credit entry in accounting) and the sinking node of the flow was affected positively by the same amount (debit entry). This means that all the asset and expense accounts in  $b$  vector have a positive balance, and liability, equity and revenue accounts have a negative balance with the total always adding to zero. This representation thus treats debit entries which correspond to increases in asset accounts or decreases in liability/equity accounts as positive and vice versa.

This network representation yields an algebraic equation (1) that relates the state at  $t+1$  to state  $t$ .

$$b^{t+1} = Eb^t + S \cdot f \quad (1)$$

The flow vector  $f$  corresponds to various monetary flows (Table I) and the effect of any single flow on the states is shown via a systems matrix  $S$ . Each column of this matrix shows a flow's positive, negative or zero effect on respective states. As an example, the flow corresponding to 'acquisition of equipment for cash' has a column in  $S$ -matrix as shown below.

$$\begin{bmatrix} 0 & 0 & \dots & -1 & \dots & +1 & \dots & +1 & \dots & +1 & 0 & -1 & \dots \end{bmatrix}' \\ \text{A LE} \dots \text{CA} \dots \text{FA} \dots \text{ME} \dots \text{BVM} \dots \text{C}$$

Note here that the cash and current asset nodes have a neg-

TABLE I.—A Non-Exhaustive List of System Flows

Flow #	Description of the Flow	Direction of Flow from/to (Cr/Db)
700	Subscription Sales - Cash	SS/C
702	Subscription Sales - Credit	SS/SR
704	Advertising Commission	C/CADC
706	Cancellation	C/SS
708	Carrier Commission	C/CC
710	Dealer Subscription - Credit	DS/SR
712	Postage Charges for Distribution by Mail	C/CMC
714	Mileage Charges for Distribution	C/VEK
716	Label Supplies - Mail Distribution	C/LS
718	Rental Charges on Equipment	AP/ER
720	Salaries and Wages - Circulation	AP/SW
722	Phone Expense - Circulation	AP/PR
724	Operating Supplies - Circulation	AP/OS
726	Mileage - Other - Circulation	C/MI
728	Travel and Entertainment - Circulation	C/TE
730	Postage - General - Circulation	C/PE
732	Stationery and Printing - Circulation	C/SP
734	Miscellaneous Expense - Circulation	C/ME
736	Contract Labor Expense - Circulation	AP/CLE
738	Rent Expense - Circulation	C/R
750	Photographic Supplies	C/PHS
752	Wire Service Rental & Supplies	AP/WSR
754	Syndicate Contract Expense	AP/SCE
756	Janitorial Expense	AP/JE

ative entry (credit entry) and book value of equipment, machinery & equipment and fixed assets have a positive entry (debit entry). All other nodes are not affected and, thus, have a zero entry. It is also important to note that the identification of a set of widely used system flows in the organization predefines the interface between states and flows through the definition of ' $S$ '. The matrix  $E$  is simply an identity matrix with the diagonal elements corresponding to states that do not accumulate between  $t$  and  $t+1$  set to zero. If the increments in  $t$  correspond to the reporting period (one year), then all the entries in  $E$  corresponding to income statement items are set to zero. This completes the description of the graphical and algebraic representation of the financial state of the system.

Decisions made at various levels in the organization affect the states of the system through a set of flows. By identifying these 'decisions centers,' one can partition the flow vector,  $f$ . This, in turn, will modularize the state vector and provide a structure for the design. The methodology used for this partitioning is discussed in [7] and will only be applied to the case discussed here. The following conventions and terminology are used for clarity in the presentation.

A decision block ( $D$ -block) is associated with a set of decisions that are considered as a unit because these are either made at a location geographically or functionally separated from others, or considered inseparable in accomplishing a specific objective that is clearly defined. We will look at each case in the next section. A  $D$ -block,  $D^i$ , may need either  $X_b^i$  and/or  $X_f^i$  for generating an optimal decision. Here  $X_b^i$  and  $X_f^i$  correspond to exogenous state and flow values needed by  $D^i$ . This is mainly determined by the decision maker in cooperation with the recording (accounting) subsystem. Identifying the flows  $f^i$  associated with  $D^i$ , one can generate sets  $W^i$ ,  $V^i$  and  $U^i$ . Here the sets  $U^i$  and  $V^i$  represent the states that are affected and exclusively affected by decision  $i$ , while  $W^i = \{U^i - V^i\}$ .

A level block ( $L$ -block) is defined as one that contains all the decisions with same level number. The level number is determined by observing the input/output interdependency among various decisions.  $X_b^i$  and  $X_f^i$  will automatically generate these level numbers using a simple algorithm [see (7)]. Conceptually the  $L$ -block identifies all the decisions that are either independent and needed together for the next level decisions, or mutually interdependent (each need inputs from the other). The structure developed in the next section will illustrate these two cases.

## THE DESIGN ALTERNATIVES

In the last section we have seen the typical financial state and the type of system flows applicable to the firm under consideration. Some aggregation of the states was introduced for clarity of the presentation and this in no way will affect the design. An example of such an aggregation is CASH which appears in many reporting documents in a more detailed form such as 'cash in bank,' 'cash in bank-payroll' and 'petty cash.' This amount of detail serves no additional purpose but to depict the realism needed for certain reporting

functions—a sub-objective of the total system. Let us now proceed with the identification of the design structure under two specific cases. These should illustrate not only the procedure used in the development of the structure but also the critical issues that played a role in the selection strategy for implementation. The implementation will be discussed in the next section.

Case 1

The first case that is considered involves the identification of decision centers as they currently exist. Even though these are not explicitly stated, the responsibility accounting adopted by the company leads us to the identification of five major decision centers—circulation, advertising, production, editorial, and administration. These centers are also geographically separated. An analysis of each center's expenditures, independent as well as overlapping, provides a clustering of nodes in the financial state as shown in Figure 1. The procedure used to generate this clustering is discussed in [8] and follows the natural sequence: activity→flow (monetary)→nodes affected.

The second step in the structuring process is the determination of the activity interdependence. For the case here there is an observable dependency between circulation and production—production requiring information on subscriber data for scheduling and for evaluating specific charges. Editorial and Advertising centers seem to operate independently of each other as do Production and Circulation except for the transfer of technical data such as where to display advertisements and how to organize the news. All the four centers feed expenditure data to the administrative center for allocating fixed charges and for determining financial charges such as interest, tax and depreciation. Using this simple flow sequence a structure is generated and is shown in Figure 2(a). Table II summarizes these observations. Note here that centers 2, 3, and 4 are flow-independent but the flows associated with these are all necessary for center 5's computations.

Conceptually the structure derived from the data in Table II is a rearrangement of the nodes in the financial state such that the activities of each decision center are mapped consistently to the financial state of the system while, at the same time, reducing the need to maintain large data matrices at each center. This feature has an appeal since its capability to reduce the information transfer among units and yet provide needed interface can go a long way in decentralizing the data base. Another important feature is the flexibility it provides for testing various organization structures with regard to their usefulness for a specific purpose such as responsibility reporting, planning and managerial reporting.

Under a broader framework each structure may be treated as a sub-schema (user's view of data) that can be managed by a data base. One may look at this as a situation where a user instead of developing a sub-schema with all the set relationships among records, which is typically the case in many DBMS systems, develops activity interdependence among decision centers. The system then generates a sub-

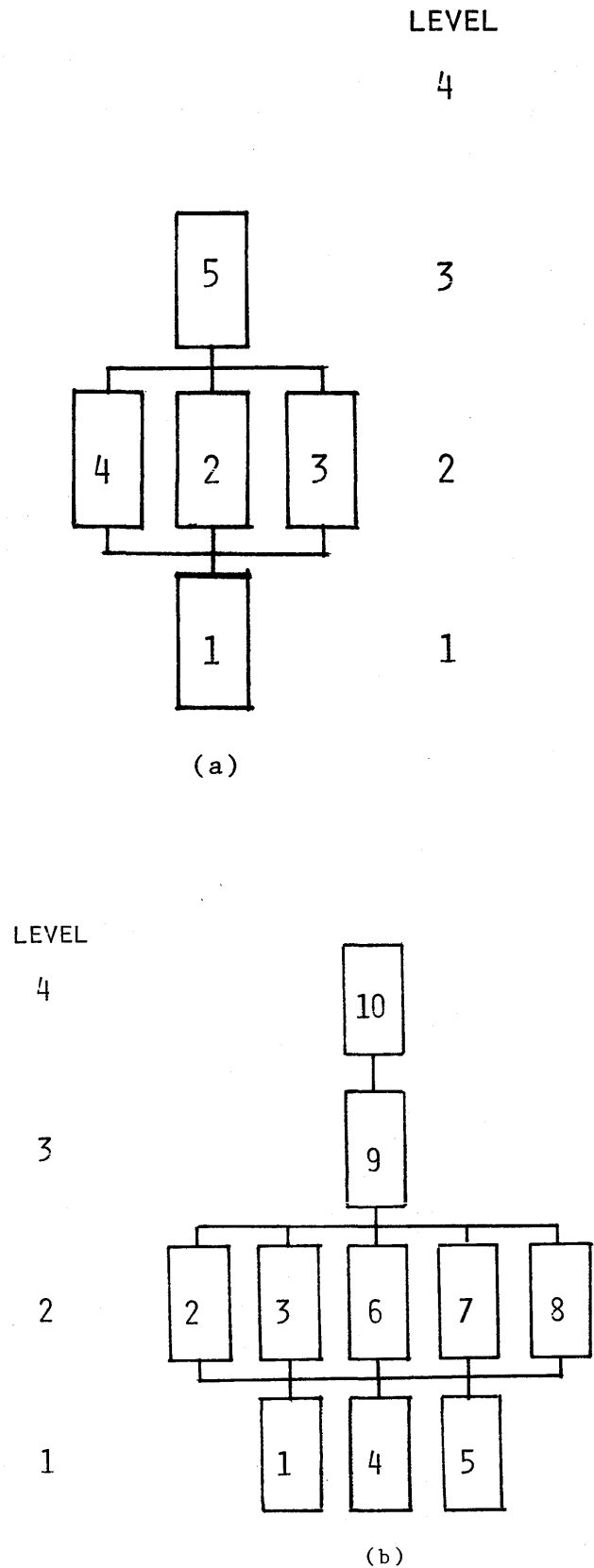


Figure 2—A decision oriented hierarchy for cases 1 and 2.

TABLE II.—Partitioning by Decision Center in Case 1

No.	Decision Center	f	v	W	$X_b$	$X_f$	Level No.
1	Circulation	700-738	SS,DS nodes in Unit 1	C,SR,AP nodes in Unit 6	--	--	1
2	Production	800-830	MS,DR nodes in Unit 2	C,AP, INV nodes in Unit 6	--	700, 702, 706	2
3	Editorial	750-780	Nodes in Unit 3	C,AP, nodes in Unit 6	--	--	2
4	Advertising	850-878	DA,CA,II nodes in Unit 4	C,AR nodes in Unit 6	--	--	2
5	Administrative	900-954	Nodes in Unit 5	Nodes in Unit 6, all leaf nodes of Balance Sheet	--	E-CIR E-PR E-ED E-ADV E-ADM	3

schema automatically for DBMS interface. This can be very useful for personnel in management who are traditionally comfortable in dealing with financial activities and their interdependence rather than record relationships. Issues related to this will not be elaborated here but will appear in future work by the author.

### Case 2

Another design strategy is to classify activities not using the organization structure currently in place (case1) but by reclassifying them for better control. Some of the problems identified in the preliminary and the detailed study of the systems were mainly due to an overlapping of activities or improper procedures used for controlling these activities. An illustration of this is when the circulation center is made responsible for distribution expenditures while production is involved significantly in the distribution activity. A greater logistic problem arises when a central data file is organized by circulation not in an order (by customer name) that best meets its objectives such as customer billing and subscription updates but in an order (by zip code) that is needed for distribution. In order to reduce the overlapping of activities

and to distribute data based on local need, a reclassification shown in Figure 3 will be used.

For this decision center classification, the activities are identified, associated monetary flows determined and a new

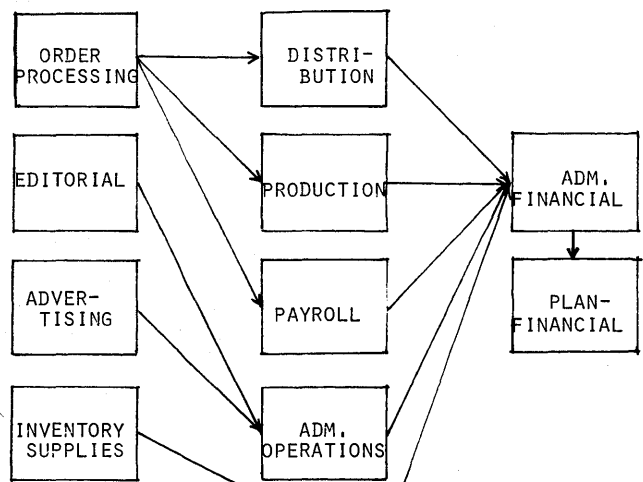


Figure 3—A decision center classification in case 2.



TABLE III.—Partitioning by Decision Center in Case 2

No.	Decision Center	f	v	W	$X_b$	$X_f$	Level No.
1	Order Processing	700, 702, 704, 706, 710	SS,DS, CADC, BSL	C,S,R	--	--	1
2	Distribution	712, 714, 716, 718	CMC,VEX, LS,ER	C,AP	--	700, 702, 706, 710	2
3	Production	800-810	MS,DR, and nodes in Unit 2	C,AP,INV	--	700, 702, 706, 710	2
4	Editorial	750-760	Nodes in Unit 3	C,AP	--	--	1
5	Advertising	850-858	DA,CA,II Nodes in Unit 4	C	--	--	1
6	Payroll	708, 720, 736, 762, 778, 812, 828, 860, 876, 900, 936, 952	CC,SW,CLE, PAT	C,AP,PWH	--	700, 702 706	2
7	Administrative Operations	722, 726 728, 734, 914, 920 and others corresponding to similar charges from each unit.	PR,MI,TE, ME,PAD,RME	C,AP	--	--	2
8	Inventory Supplies	724, 732, 730 and similar costs from each other unit.	OS,PE,SP, INV,BME, BFF,BAT	AP,C	--	--	2
9	Administrative-Financial	738, 780, 830, 878, 954, 902 ... (except 914, 920) ....1000	R, all nodes in Unit 5 except PAT, PAD and RME, and all other leaf nodes.	Leaf nodes of B/S items.	--	724, 730 732, ... and all others	3
10	Planning-Financial	1001-1050	CS,PS,APC, DIV,LL,CA	CA	CA,OA, FA,CNI	--	4

structure is developed. This structure is illustrated in Figure 2(b) and the relevant data is presented in Table III. Note that the level numbers for centers 4 and 5 can be one instead of two as assigned here. The selection here is based on computational convenience.

The objective of the study was to implement this structure and to evaluate the benefits derived using this approach. Note that the selection of this strategy is primarily based on the relative importance of the data for cost accountability and reporting needs. Other structures can be evaluated based on the overall objectives of the firm and, as mentioned earlier, this methodology for structure design facilitates quick testing of each structure. The next section will discuss some of the procedures that will be followed in the implementation.

THE IMPLEMENTATION

The decision center classification and the associated partitioning of the flow vector led us to a structure illustrated in Figure 2(b). This structure consists of four *L*-blocks and ten *D*-blocks. This section will show how this information can be used to create a data base that can then be effectively managed either by the use of DBMS system or by a file management system. Let us first identify all the information that needs to be maintained thus far about the structure.

The decision block *i* at level *j* is related to the sub-schema at that level according to equation (2).

$$b_j = A_2 b_{L-v,j} + \sum_{i \in I_j} A_1^i b_{i^j} \tag{2}$$

Here,  $b_x$  is the vector of values associated with set *x*, *L* is the set of leaf nodes of the schema (Figure 1),  $A_1$  and  $A_2$  are the appropriate aggregation matrices, and  $I_j$  corresponds to the number of decision centers in level *j*. For example, the *L*-block two contains five *D*-blocks: 2,3,6,7, and 8. See [7] for more details on this mapping procedure.

To properly relate the set of *D*-blocks in one level with another through a set of sub-aggregation points, a top-down design is used. The *L*-block with the largest level number is designed first and then mapping is done with the *D*-blocks of the next level. Equation (3) shows the mapping between levels *i* and *i*-1. The

$$b_{L-v,i} = E_{i-1} b_{i-1} \tag{3}$$

identity matrix *E* has zero entries on the diagonal elements associated with those states (sub-aggregation points) that do not transfer any valuable information from level *i*-1 to *i*. These states are shown in parentheses in Table IV along with the breakdown of nodes at each level.

Due to certain resource limitations, only a simple file management system will be designed for implementing this structure. At the end, however, it will be shown how a network structure can be created from this information for possible DBMS use. It is apparent at this stage that one needs to maintain an output file for each level, a file for each *D*-block, a process for each level to do the necessary algebraic mapping and a process for each decision. (See Figure 4.)

Each *L*-block *i*, say, is associated with an output file that contains information on  $b_i$  at that level which is used by the higher level (*i*+1) process to compute  $b_{L-v,i+1}$  using Equation 3. It also contains either *f* and *v*, or leads to *f* and *v* that are generated by the *D*-blocks of that level. Here *L*3 output file will then contain either the values of the system flows and non-monetary variables determined by the decision center 9 ( $f^9, v^9$ ) or the name of file(s) that contains them. The process block associated with level *i* also performs the mapping of flows in level *i* to  $b_i$  using  $b_{L-v,i}$  and  $b_{i^j}$  according to Equation 2. Note that  $b_{i^j} = S^j f^j$  where  $S^j$ , a partition of *S*, is induced by the partitioning of *f*.

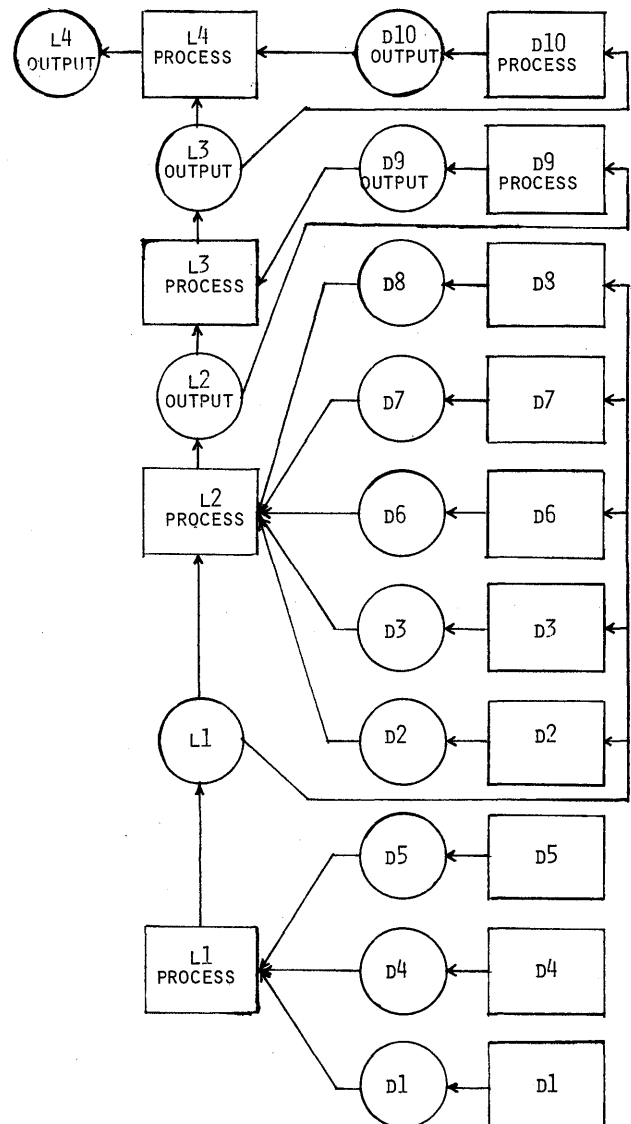


Figure 4—A mapping structure for the implementation of case 2.

TABLE IV.—Inter-Level Mapping in Case 2

Level	Decision Center	U*	L-V	b
4	10	<u>DIV</u> , <u>CS</u> , <u>PS</u> , <u>APC</u> , <u>LL</u> , <u>CA</u>	CA, OA, FA, CL, CNI	CA, OA, FA, CL, LL, CNI, DIV, CS, PS, APC, A, L, LE, E, RE, CE
3	9	<u>C</u> , <u>AR</u> , <u>PPI</u> , <u>NR</u> , <u>SR</u> , <u>D</u> , <u>ASL</u> , <u>DME</u> , <u>DFE</u> , <u>DAT</u> , <u>AP</u> , <u>NP</u> , <u>STL</u> , <u>PWH</u> , <u>AEX</u> , <u>USR</u> , <u>R</u> , LL, all nodes in Unit 5 except PAT, PAD, and RME	BSL, INV, BME, BFF, BAT, PAT, PAD, RME, notes in Unit 6 except R, leaf nodes of S, E-CIR, E-PR, E-ED, E-ADV.	CA, OA, FA, CL, CNI, (OPE, SL, ME, FF, AT, all leaf nodes of Figure 1)
2	2, 3, 6, 7, & 8.	<u>MS</u> , <u>DR</u> , <u>PAT</u> , <u>PAD</u> , <u>RME</u> , <u>INV</u> , <u>C</u> , <u>AP</u> , <u>PWH</u> , <u>BME</u> , <u>BFF</u> , <u>BAT</u> , nodes in <u>unit 1, 2, and 6</u> (Except R)	SS, DS, DA, CA, II, BSL, C, SR, AP, nodes in units 3 and 4	(E-CIR, ER-PR, S)
1	1, 4, & 5.	<u>SS</u> , <u>DS</u> , <u>CADC</u> , <u>BSL</u> , <u>DA</u> , <u>CA</u> , <u>II</u> , nodes in units 3 and 4, C, SR, AR	--	(E-ADV, E-ED)

\*Underlined node set in U corresponds to set V.

The output file associated with each *D*-block has all the output information of that decision center, both monetary and non-monetary. The process associated with a decision center contains model(s) used to compute *f*. The exogenous input they need ( $X_b, X_f, X_v$ ) will come either from or through the output file of the next lower level. Certain procedures to structure the modelling activity, if linear models are used to compute *f* (this is feasible in the case of many accounting functions such as cost allocations, interest and depreciation computations, and payment and collection procedures), are illustrated in [9] and will be used appropriately during the implementation.

The identification of the mappings and the associated file management can be automated once the system designer identifies the flow partitioning, the exogenous flow or variable data needed for making each decision, and flow com-

putation (model definition) at each decision center. If a DBMS environment is used, the relationships can be identified as shown in Figure 5.

One can define this using a network data base. Note here that all the set relationships and the instances of each record type associated with a strategy can be automatically derived simply from a users definition of activity/flow interdependence. This should facilitate a user to define the data instances in a language that is familiar to him/her.

At this stage, the actual implementation is not complete and, hence, the benefits of, or difficulties in, the design of systems in this manner cannot be evaluated objectively. It is hoped that the modularity provided in this approach will facilitate a step-by-step approach to the integrated system design and some of the observations in this regard will be made in another paper.

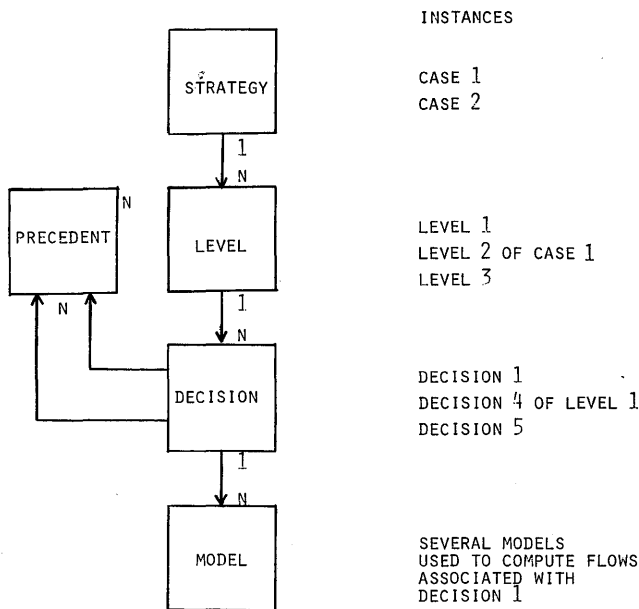


Figure 5—A data base schema for storing the planning data.

## CONCLUSIONS

In this paper, we have seen how a financial state of an organization, expressed in terms of chart of accounts and flows affecting these accounts, can be partitioned into manageable modules (levels and decisions) by using the concept of decision center identification. With this modularized structure a consistent mapping among modules both among levels and within levels can be automatically generated. Each module can then be designed independently so as to meet the specific objectives of that module along with the overall objectives of the organization that are expressed in terms of the flows originating from this decision center. It is shown that this structured approach to designing a system not only facilitates a step-by-step design but also makes the decentralization of the total data base feasible.

"Integrate Now" approach often comes under criticism since the identification of a large data bank (data and relations) that serves various users is difficult, and the implementation of such a data base to meet the uncertain and dynamic processing requirements of these users is expensive. By using this top-down structured approach, the difficulty of data bank identification is reduced since the integration is first limited to the financial flows that are well-defined and crucial to the success of an organization. By designing each module independently with the knowledge of the extent of its interface to the total system, one should

be able to decentralize the data base to meet the specific needs of the decision center associated with each module. This decentralization becomes important as the trend continues toward the use of mini-computers and micro-processors to meet the specific needs (ex: advanced modelling capabilities) of some decision centers. Since most of the financial reporting and planning activities of an organization are centralized and controlled by top-management, the approach discussed here can transfer only the needed information for modules to the center and still provide for a consistent, modular, and possibly an inexpensive integrated system.

## SUMMARY

The paper illustrates a top-down design of an information system for newspaper industry. The basic goals of the organization are used to structure the design process such that a comprehensive system can be designed to support both planning and operational activities. The modularity provided by this structure facilitates not only a step-by-step approach to the actual implementation of the system but also facilitates distributed data management that is found to be convenient for the case under consideration. The integration of the activities is provided by the use of various accounting transaction types that appear in the normal reporting process. Some other benefits of representing and designing the system in this manner are also discussed.

## REFERENCES

1. Donaldson, James R., "Structured Programming," *Datamation*, December 1973.
2. Langefors, B., "Some Approaches to the Theory of Information Systems," *BIT* 3, 1963, p. 229-254.
3. Teichrow, D., "Problem Statement Languages in MIS," *Proceedings, International Symposium of BIFOA*, Cologne, July, 1970, pp. 253-270.
4. Nunamaker, Jr., J. F., Konsynski, Jr., B. R., Ho, Thomas, and Singer, Carl, "Computer-Aided Analysis and Design of Information Systems," *Comm. of ACM*, Vol. 19, No. 12, 1976, pp. 674-687.
5. Tanniru, M., "A Decision Support System for Planning," Ph.D. Dissertation, Northwestern University, 1978.
6. Blim, J. M., Stohr, E. A., and Tanniru, M., "Design of a Corporate Information System," *Proceedings of IEEE Conference*, Chicago, November, 1977.
7. Tanniru, M., "A Structured Information System for Planning," presented at the *First International Symposium on Policy Analysis and Information Systems*, and will appear in its proceedings (1979).
8. Stohr, E. A. and Tanniru, M., "The Design of a Corporate Planning System Simulator," presented at the *ACM Winter Simulation Conference-78*, Miami and appeared in the Conference Proceedings.
9. Blin, J., Stohr, E. A., and Tanniru, M., "A Structure for Computer Aided Corporate Planning," *Policy Analysis and Information Systems*, Vol. 2, No. 2, December 1978.



# SID: a system for interactive design

by TOSIYASU L. KUNII and MINORU HARADA

The University of Tokyo  
Hongo, Tokyo, Japan

## INTRODUCTION

A System for Interactive Design (SID) is a computer-aided visual facility for hierarchical (or recursive) design of complex systems. SID is built as a provision to make the potential of our graph theoretical design tool RGF (the recursive graph formalism) actually available to system designers. RGF,<sup>1,2,3</sup> as we initially proposed in 1978, aimed at providing a logical basis for interactive design evolution from global to detailed, and/or from simple to complex.<sup>4,5</sup> RGF was actually applied to designs of hospital information systems<sup>1</sup> and petrochemical plants,<sup>3</sup> and was proven useful for logically detecting and preventing human design errors and for computer-aided design evolution. SID includes the capabilities of SARA of UCLA<sup>6</sup> and SADT of SofTech<sup>4</sup> which are known as the system specification methodologies based on hierarchically structured graphs. Related works using similar graphs in other areas are H-graphs<sup>7</sup> in language and automata theory, and DRLH<sup>8</sup> in artificial intelligence.

Basically, SID consists of an interactive computer graphics to display design specifications for designer's visual inspection, a database system to verify, store, update, retrieve and control the design specifications in a shared file, and a design processor to execute design operations. In the databases, both design specifications and design operators are stored to allow sharing of frequently used designs<sup>1,3</sup> and design processes<sup>2</sup> as well. Another major task of the current version of SID is to provide system developers with design evaluation facilities. The motivation for this is to cut down the system development cost by precluding the chances of implementing "poor" (i.e., would be marked "poor" if evaluated) designs.

## DESIGN OF SYSTEM STRUCTURES AND PROPERTIES

Any system design usually consists of the specifications of the structures and properties of a system. The *structure* describes the system organization representing its subsystems (also called components, parts, or elements), environments, interfaces (also called ports, gates or terminals) and their relationships. The *properties* describe various information, associated with the system and its structure. A graphical representation of the design specification is called

a *design schema*. The system components, environments and interfaces can have their own structures and properties, and hence, for the sake of generality, can be recursively viewed as systems again.

As a matter of fact, designer's recursive view can go two ways—from general to special and from special to general. More precisely stated, when the structure of the system is designed, any other systems related with it are classified into three system categories depending on their relationship types: (1) system category "environment," when the relationships are *associations*—that is, the systems are outside a given system; (2) system category "subsystem," when the relationships are *inclusions*—that is, the systems are inside a given system; (3) system category "port," when the relationships are *interfaces*—that is, the systems are on the boundary of a given system.

Once the designing of the structure of the system is done, the structure can be related with various records of information in a database as its properties. As usual in any database, each record consists of several fields. Depending on whether a record is related with a system or a relationship, it is called a system record or a relationship record. Therefore, the design of the system properties is a simple matter of relating the system structure with the database records.

## RECURSIVE GRAPH FORMALISM (RGF)

The *recursive graph formalism* (RGF) is devised to combine the following two major potentials of widely used interactive design tools into one: (1) design visuality by displaying graphs for easy human understanding and inspection of hierarchical design evolution as typical in hierarchical structure diagrams and system charts<sup>6,7,9</sup>; (2) design automation by machine processing, analysis and evolution of formally specified design as typical in various hardware design<sup>10</sup> and program flow analysis methods.<sup>11</sup>

RGF consists of recursive graphs (*R-graphs*) to represent design specifications, and recursive graph operators (*R-operators*) to manipulate them.

### *R-graphs*

Traditionally, the most popular way of representing structures is by a graph, because it is both visual and formal. A

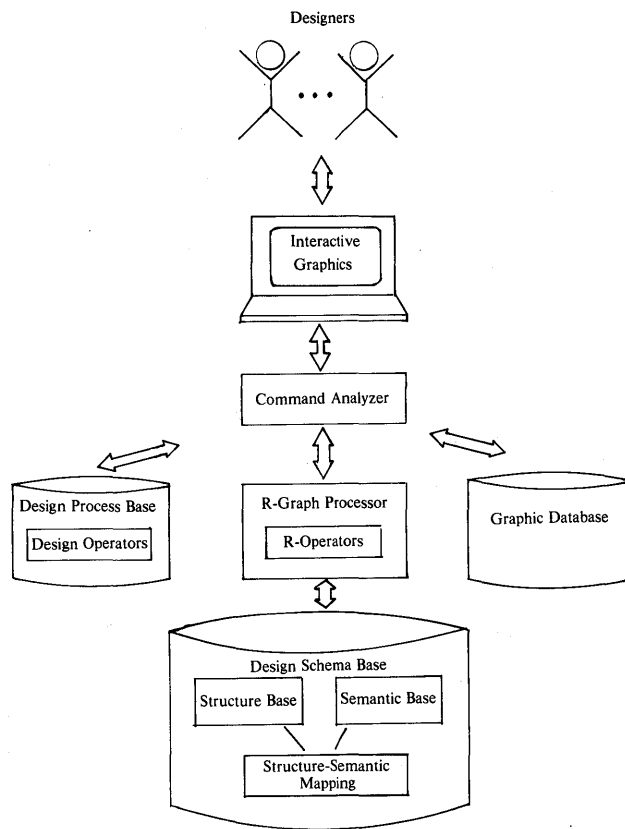


Figure 1—SID architecture.

graph  $G$  is simply a triple which consists of nodes  $N$ , arcs  $A$ , and an arc function  $af$  specifying the ordered node pairs to which arcs are incident. Then,

$$G = (N, A, af),$$

where  $af: A \rightarrow N \times N$ .

Usually systems are represented by nodes, and their relationships by arcs. To utilize all the powers of mathematics, formalism and algorithms of graph theory, we extend graph theory to incorporate designer's recursive view of the system structures and of the associated system records. First, an arc function  $af$  is extended to map an arc to a pair of node subsets such that  $af: A \rightarrow 2^N \times 2^N$ . This extension is useful for a designer to relate groups of nodes by an arc. Next, a *subnode function* ( $sn: N \rightarrow 2^N$ ), a *port function* ( $pt: N \rightarrow 2^N$ ), and a *subarc function* ( $sa: A \rightarrow 2^A$ ) are annexed to incorporate inclusion relationships among systems, interface relationships among systems, and inclusion relationships among associations, respectively. These extensions increase the structure representation capabilities of graphs. Further extension is done to give the property representation capabilities to graphs. That is, a *node semantic function* ( $ns: N \rightarrow NR$ ) and an *arc semantic function* ( $as: A \rightarrow AR$ ) are annexed to link nodes and arcs to node records  $NR$  and arc records  $AR$ , respectively. Thus, an R-graph is given by:

$$R = (N, A, af, sn, pt, sa, ns, as).$$

The domains of the functions  $af$ ,  $sn$ ,  $pt$ ,  $sa$ ,  $ns$  and  $as$  are extended to incorporate the value "undefined (or under defined)" and the value "overdefined (or redundant)."<sup>12</sup> The value "undefined" indicates that "the value is not available now but will come." It is different from the "null" value which means "the fact that no value exists is made sure."

Given a design schema, repeated applications of  $sn$  and  $pt$  to the nodes and of  $sa$  to the arcs produce a *hierarchy* of the nodes and that of the arcs, respectively. We call them a node hierarchy and an arc hierarchy. Given a node or an arc of any hierarchy, the nodes or arcs produced earlier are called its *ancestor* nodes or arcs. It is reasonable to assume that in actual system design, any system cannot be subsystems (or ports) of two different other systems at the same time. Hence, in the rest of this paper, we only consider the case where the node hierarchies and the arc hierarchies form DAG (directed acyclic graphs). This actually increases the logical clarity of our formalism.

### Design schema base

Two groups of *records*, one representing the structures and the other representing the properties of a system, are stored in a design schema base separately and as *flat tables*, so that both can be freely combined and utilized depending on application views.<sup>5,13</sup> As a matter of fact, to substantiate this flexibility of table forms, we applied a *pointer array* technique to implement a database mapping. Figure 2 illustrates how a given R-graph is stored in a *design schema base*. In the *structure base*, node identifiers  $NID$  and arc identifiers  $AID$  are used as the primary keys of the tables which define functions  $af$ ,  $sn$ ,  $pt$  and  $sa$ . In the *semantic base*, tuple identifiers  $TID$  are the primary keys of node record  $NR$  and arc record  $AR$ . We now can define the structure-semantic mappings,  $ns$  and  $sa$ , as pointer arrays using only these identifiers. Hence the structure base and the semantic base can be updated as independently as possible. Please note that the meaning of symbols in Figure 2 are found in Figures 3 and 4.

### R-operators

All the elementary operations to a node or an arc of a design schema are performed by recursive graph operators (R-operators). Such most primitive operations are collectively called an *R-graph processor*. Since design schemas are represented as flat tables, actually a *table handler* can be used as an R-graph processor. Throughout this paper, whenever necessary, R-operators are defined each time and used rather intuitively to increase the readability.

## DESIGN PROCESS FORMALIZATION FOR DESIGN PROCESS SHARING

Design processes often contain similar and/or common basic processes, for example, to produce, integrate, reduce,

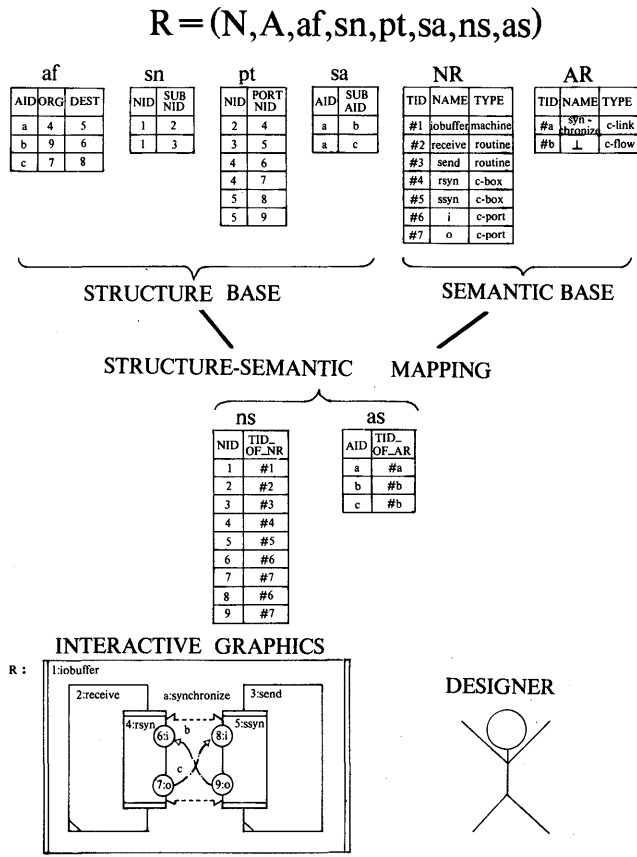


Figure 2—Recursive graph components.

analyze and verify a design schema. In SID, a design course of a system interactively specifies a shareable design process as a sequence of R-operators. We call such a shareable design process also a *design operator*. Actually, these processes are stored in a design process base for a designer's later use and/or for sharing them among designers. They are driven through an interactive design command, issued by a designer. Combined with the design schema base, this design process base provides a flexible and powerful graphics-oriented tool for designers.

In the rest of this paper, we show examples of design processes to be shared. We also show that these design processes are actually representable in a common formalism of R-operators. For illustration, a well-known case of designing a type of *concurrent system*, a pipeline system, is examined.

A *pipeline system* is a system which inputs a text from a card reader, formats it, and outputs it to a line printer.<sup>14</sup> The text is formatted so that each text begins and ends with a blank page, each page begins and ends with a blank line, and each line is surrounded by blank margins. There, inputting, formatting and outputting a text are processed concurrently. The global architectural design of the system is represented in Figure 5. The symbols used in this figure are explained in Figures 3 and 4. "Type Name" lists the reserved types of systems or of associations, "Graphic Symbol" is for display, and "Meaning" is for annotation.

	Type Name	Graphic Symbol	Meaning
Macro Active System	Machine		A machine represents a system which performs multiple functions and consists of routines performing those functions and data for communication among those routines.
	Routine		A routine represents a system which performs a single function.
	Communication box		A communication box represents a structured interface for communication among routines.
Primitive Active System	Routinecall		A routinecall represents an invoking of a routine.
	Assignment		An assignment represents a sequence of expression evaluations and data transfers.
	Case		A case represents a multiway control branch by a selector expression evaluation.
	Token holder		A token holder represents a place in Petri net model.
Passive System	Control port		A control port is a port for control entry and exit of a macro active system.
	Fork, Join		A fork and a join represent a transition having only one incoming and one outgoing c-flow, respectively, in Petri net model.
	Data storage		A data storage represents a system holding some data.
Passive System	Parameter		A parameter represents a parameter of a machine or a routine.
	Access channel		An access channel represents a place through which a data is transferred between the inside and the outside of a macro active system.

Figure 3—Reserved system types for concurrent system design.

	Type Name	Graphic Symbol	Meaning
Primitive association	Data-flow (abbr. d-flow)		Primitive active system P reads data D1 and writes data D2.
	Control-flow (abbr. c-flow)		Activation of primitive active system P1 precedes that of primitive active system P2.
	Routinecall-flow (abbr. r-flow)		Routine R defined at the destination of a routinecall-flow is invoked at the origin of the flow.
Macro association	Control-link (abbr. c-link)		A control-link indicates existence of control-flows from ports of communication box C1 to those of communication box C2.
	Use-link (abbr. u-link)		A use-link indicates existence of routinecall-flows from the inside of routine R1 to routine R2, and represents that routine R1 uses routine R2.
	Hierarchy-link (abbr. h-link)		A hierarchy-link indicates existence of use-links from routine R or the inside of machine M1 to the inside of machine M2, and represents that M1 and R are implemented by using routines of M2.

Figure 4—Reserved association types for concurrent system design.



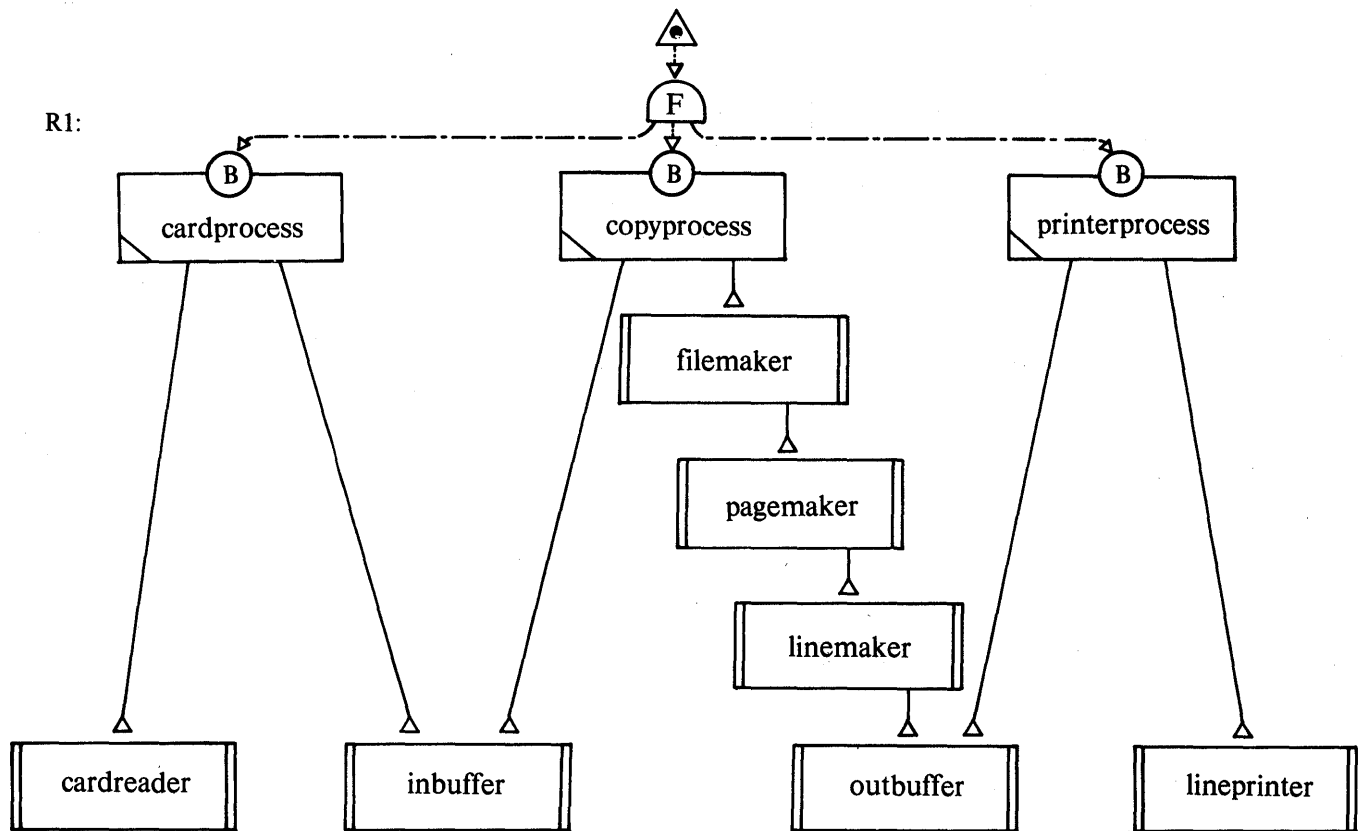


Figure 5—A design schema of the global architecture of a pipeline system.

*Design integration/reduction operators*

Here, very often used design processes for *hierarchical design evolution*, such as design integration, design abstraction and design reduction, are considered. We show that they actually can be defined in a form, applicable to general design schemas, and hence shareable among the designers.

Suppose a designer designs an input/output buffer “iobuffer.” It is intended to be used as a common design schema of both the “inbuffer” and the “outbuffer” subsystems of a pipeline system which is shown in Figure 5. Figure 6 contains design schemas R2-R6 used in the following discussions. They are all specified in a shareable design form of R-graphs and stored in a design schema base. The designer starts to identify, as the subsystems of “iobuffer,” the shared data storage “contents” and two routines “receive” and “send” which operate in parallel, communicating through the c-link “synchronize” as shown in design schema R2. Next, the designer refines “receive,” by specifying its data flow as shown in R3 and also its control flow as shown in R4. Thus, the phase of designing the “iobuffer” subsystems is completed.

Now the design gets into the phase of subsystems integration. First, the designer produces the fully specified design schema of “receive” as R5. Actually, this design process is defined as an operation of *integrating* R3 and R4 by

merging their common parts. In this case, the parts to be merged are two node pairs. One pair is (“receive” in R3, “receive” in R4). The other pair is (“text: = contents” in R3, “text: = contents” in R4). We specify the integrating process as a *JOIN operation*:

$$R5 = JOIN(R3, R4), \{ (“receive” \text{ in } R3, “receive” \text{ in } R4), (“text: = contents” \text{ in } R3, “text: = contents” \text{ in } R4)\}, \phi,$$

where  $\phi$  denotes an empty set (in this case, of arc pairs). This is just an instance of a general form

$$r1 = JOIN(r2, r3, SN, SA),$$

where  $r2$  and  $r3$  are design schemas to be joined,  $r1$  is the resulting design schema, and  $SN$  and  $SA$  are lists of node pairs (including ports pairs) and arc pairs, respectively, to be merged. Since general forms are obvious from their instances, only instances are shown in the following discussions.

Next, the designer inserts schema R5 into the “receive” node of R2 by matching port pairs (“rsyn” in R2, “rsyn” in R5) and (“contents” in R2, “contents” in R5), and produces R6. This process is also defined as a *ZIN (zoom-in*

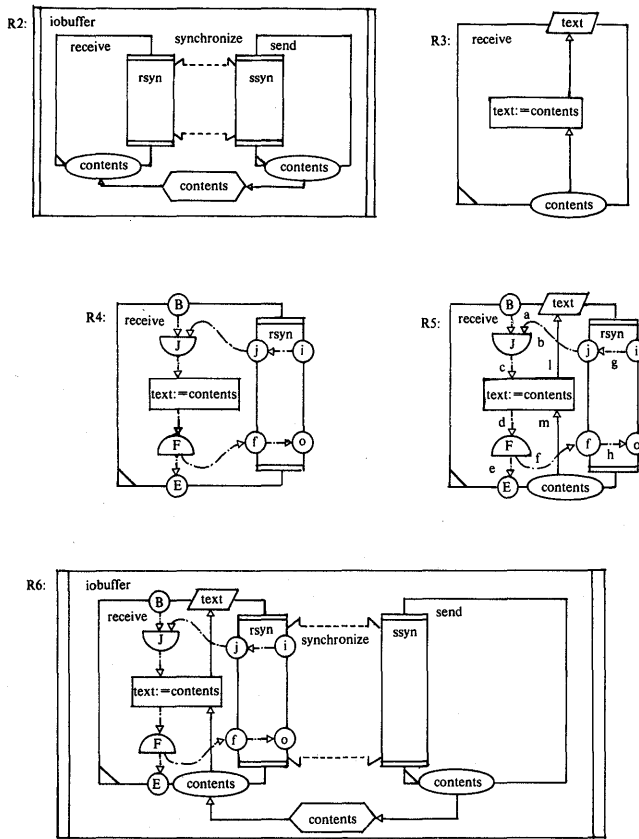


Figure 6—Design schemas appearing during the iobuffer design process.

node) operation as follows:

$$R6 = ZIN(R2, R5, \text{"receive" in R2, \{("rsyn" in R2, "rsyn" in R5), ("contents" in R2, "contents" in R5)\}})$$

This completes the subsystems integration phase. So far, we have identified two general integration operators, JOIN and ZIN.

We now consider a case where the designer likes to see an abstract (i.e., global) structure of a given schema. For example, in schema R6, the designer wishes to hide the uninterested details of "receive." This is done by applying ZON (zoom-out node) operation to R6 to obtain more global schema R2:

$$R2 = ZON(R6, \text{"receive"});$$

conversely, the designer can extract the interested inside detail of "receive" by EXN (extract node) operation:

$$R5 = EXN(R6, \text{"receive"}).$$

Furthermore, if the designer wishes to analyze R5 from the view point of data flows, he or she uses SEL (select) operation for gathering the related nodes {"receive", "text", "text:=contents", "contents"} and arcs {"l", "m"}, and

gets R3:

$$R3 = SEL(R5, \{\text{"receive"}, \text{"text"}, \text{"text:=contents"}, \text{"contents"}\}, \{\text{"l"}, \text{"m"}\}),$$

or uses DEL (delete) operation for removing some nodes and arcs as the unnecessary details of R5:

$$R3 = DEL(R5, \{\text{"B"}, \text{"E"}, \text{"rsyn"}, \text{"i"}, \text{"o"}, \text{"j"}, \text{"f"}, \text{"J"}, \text{"F"}, \{\text{"a"}, \text{"b"}, \text{"c"}, \text{"d"}, \text{"e"}, \text{"f"}, \text{"g"}, \text{"h"}\}).$$

### Design survey operators

A survey operator is a typical example to see how a design operator can be implemented by using R-operators and other design operators. For the detailed definition of the survey operator as a sequence of R-operators, please refer to Appendix 2.

A survey operator produces a design subschema by surveying a given design schema from a specific point of view. We now define a general survey operator SURV(R, NC, AC, SC) to search any given design schema R so that a designer can find such portions of R that satisfy his or her point of view. A general way of specifying a point of view is as a set of search conditions. In this case, they are NC, AC and SC. We give here a little bit of terminological preparations to explain search conditions. A set variable is a variable which takes a subset of values. A variable to represent a field of a database record is an example of it. A field name variable is a variable which takes a field name as its value. We are now ready for explaining search conditions. The first search condition NC, called a node search condition, is in a form  $NC_1 \wedge \dots \wedge NC_m$  where  $NC_i = (NKEY_i \text{ of } NFIELD_i)$ .  $NKEY_i$  is a set variable, and  $NFIELD_i$  is a field name variable.  $NC_i$  indicates to select nodes if and only if their record contains one of the search key  $NKEY_i$  in the field  $NFIELD_i$ . The second search condition AC, called an arc search condition, is defined in the same way with NC, by using field name variables  $\{AFIELD_1, \dots, AFIELD_n\}$  and set variables  $\{AKEY_1, \dots, AKEY_n\}$ . The third search condition SC, called a structure search condition, is in a form (struct by SKEY), such that a set variable SKEYC {"NODE", "ARC"}. "NODE" and "ARC" are literal constants, which, if appear in a given SKEY, indicate to select all the ancestor nodes of nodes satisfying the search condition NC in the node hierarchy, and to select all the ancestor arcs in the same way for AC, respectively.

Examples of quite commonly taken viewpoints are a "flow of control" and a "flow of data." As search conditions, we need to use only arc search conditions with "c-flow" for a control flow, and "d-flow" for a data flow. Thus, the data flow R3 and the control flow R4 of a given design schema R5 in Figure 6 are generated by simply applying SURV to R5 as follows:

$$R3 = SURV(R5, \{ \text{"d-flow"} \} \text{ of "TYPE"}), (\text{struct by } \{\text{"NODE"}\}),$$

$$R4 = SURV(R5, \{ \text{"c-flow"} \} \text{ of "TYPE"}), (\text{struct by } \{\text{"NODE"}\}),$$

where node conditions are null.

*Design analysis and evaluation operators*

Design analysis and evaluation operators are key tools which designers need to use in management of system development. In principle, application of them directly to a given design schema gives us the results of design analysis and evaluation. Experiences of systems development tell us that modularity is one of the most important common factors when evaluating various designs. The better the modularity of a design, the smaller the range of *ripple effects* that the modification of the design causes. Therefore, when maintaining or adapting the design to application changes, good modularity certainly localizes the ripple effects of design modification, thus decreasing the *maintenance cost*. Early analysis and evaluation of a design by modularity or any other measures have another significant managerial meaning. As mentioned before, it also certainly decreases the life cycle cost of the system being developed. This is because system developers can avoid implementing designs which are evaluated as "poor." As a case study, we illustrate in the following the use of SID, especially RGF, for defining a modularity analysis and evaluation operator.

**Modularity**

*Modularity* of a set of systems defined by Myers<sup>15</sup> is known as an effective measure of ripple effects when one of the systems is modified. We show here that his definition can be formulated based on RGF and then can be evaluated automatically. As an example, we apply the results to evaluate the modularity of R7 in Figure 7. R7 is the design schema of the formatting subsystem of a pipeline system.

Suppose systems are connected only through shared data. Such *data sharing* is classified into three types specified by the following three predicates. Note that in the rest of this paper "the system represented by node *x*" is simply called "system *x*", and "*A* is defined by *B*" is denoted as "*A*  $\triangleq$  *B*"; (1) share (*x,y*)  $\triangleq$  system *x* and system *y* have access to common data *d*; (2) consist (*x,y*)  $\triangleq$  system *y* has access to data *d* which is a component of system *x*; (3) pass (*x,y*)  $\triangleq$  system *x* invokes system *y* passing at least one parameter. The above predicates are formulated in Appendix 1 based on RGF.

*Modularity operator* MDL(*R*) for design schema *R* first identifies all these relationships among systems {*x<sub>i</sub>*} such that *x<sub>i</sub>*  $\in$  *N* and (stype(*x<sub>i</sub>*) = "machine" or "routine"), and represents them in *n* × *n* square *connectivity matrix* *C* = (*C<sub>ij</sub>*), where *i*th column and *i*th row represent system *x<sub>i</sub>*, and *n* is the number of systems {*x<sub>i</sub>*}. Each matrix entry *C<sub>ij</sub>* is defined such that

$$C_{ij} \triangleq t_{ij} + s_{ij}S + c_{ij}C + p_{ij}P$$

where  $t_{ij} \triangleq$  if  $i=j$  then 1 else 0;  
 $s_{ij} \triangleq$  if share( $x_i, x_j$ ) then 1 else 0;  
 $c_{ij} \triangleq$  if consist( $x_i, x_j$ ) or consist( $x_j, x_i$ ) then 1 else 0;  
 $p_{ij} \triangleq$  if pass( $x_i, x_j$ ) or pass( $x_j, x_i$ ) then 1 else 0.

In the above, *S*, *C* and *P* are the probabilities that the mod-

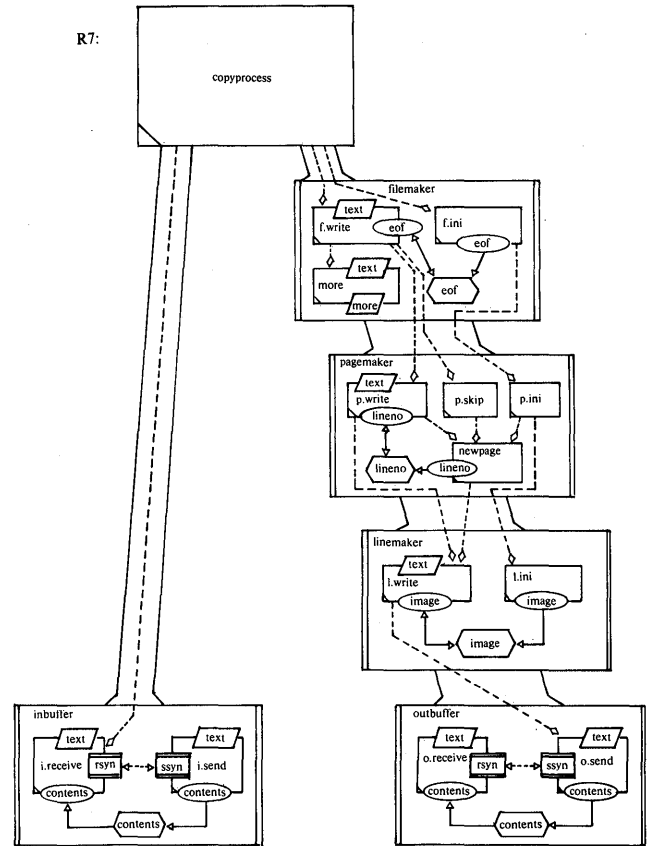


Figure 7—A design schema representing Hansen's design of the formatting subsystem of a pipeline system.

ification of system *x* causes the modification of system *y* which is connected to system *x* through data sharing specified by share, consist and pass predicates, respectively. Myers' estimated values of these probabilities are as follows: *S* = 0.7, *C* = 0.6 and *P* = 0.2.

For example, the connectivity matrix of R7 is presented in Figure 8. Using such a connectivity matrix, modularity MDL(*R*) is defined such that

$$MDL(R) = (\sum_{i \in [1, n]} \sum_{j \in [1, n]} C_{ij}) / n.$$

MDL(*R*) means the expectation of the number of the systems which must be modified when one of the systems {*x<sub>i</sub>*} is modified. Now, MDL(R7) is evaluated by using Myers' values as follows:

$$MDL(R7) = (17 + 10S + 12C + 14P) / 17 = 1 + 0.59S + 0.71C + 0.82P = 2.0$$

**CONCLUSION**

The capabilities of a System for Interactive Design, SID, currently tested at the University of Tokyo, were demonstrated. Their features are summarized in two points: "logical exactness"—"flexibility" combination, and "design"—

	i.receive	i.send	copyprocess	filemaker	f.write	f.ini	more	pagemaker	p.write	p.skip	p.ini	newpage	linemaker	l.write	l.ini	o.receive	o.send
i.receive	1	S	P														
i.send	S	1															
copyprocess	P		1		P												
filemaker				1	C	C											
f.write			P	C	1	S	P		P								
f.ini				C	S	1											
more					P		1										
pagemaker								1	C			C					
p.write					P			C	1			S		P			
p.skip										1							
p.ini											1						
newpage								C	S			1		P			
linemaker													1	C	C		
l.write									P			P	C	1	S		P
l.ini													C	S	1		
o.receive																1	S
o.send														P	S	1	

Figure 8—Connectivity matrix of Hansen’s design of the formatting subsystem of a pipeline system.

“design process” sharing. Shared design processes, called design operators, included design integration, abstraction, analysis and evaluation operators. It was a realization of our recursive design methodology RGF based on an extended graph theory. For illustration, some results of its applications to concurrent system design were also given.

ACKNOWLEDGMENT

The support of the JIPDEC, under Grant “5th Generation Computers,” is gratefully acknowledged. Further support was provided by the MESC Software Engineering Project. Authors also wish to thank Miss M. Onoda, Dr. N. Ohbo, Messers. H. Kitagawa and A. Urabe for help in preparing this paper.

REFERENCES

1. Harada, M., Kunii, T. L., and Saito, M., “RGT: The Recursive Graph Theory as a Theoretical Basis of a System Design Tool DESIGN-TOOL—With an Application to Medical Information System Design—,” *Proceedings of the International Symposium on Medical Information System*, Osaka, Japan, October 1978, pp. 503-507.
2. Harada, M. and Kunii, T. L., “A Design Process Formalization,” *Proceedings of the IEEE Computer Society’s Third International Computer Software and Applications Conference*, Chicago, November 1979, pp. 367-373.
3. Buchmann, A. P. and Kunii, T. L., “Evolutionary Drawing Formalization in an Engineering Database Environment,” *Proceedings of the IEEE Computer Society’s Third International Computer Software and Applications Conference*, Chicago, November 1979, pp. 732-737.
4. Ross, D. T., “Structured Analysis (SA): A Language for Communicating Ideas,” *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, 1977, pp. 16-34.

5. Kunii, T. L., Weyl, S., and Tenenbaum, J. M., “A Relational Data Base Schema for Describing Complex Pictures with Color and Texture,” *Proceedings of the Second International Joint Conference on Pattern Recognition*, Lyngby-Copenhagen, August 1974, pp. 310-316.
6. Campos, I. M. and Estrin, G., “Concurrent Software System Design Supported by SARA at the Age of One,” *Proceedings of the 3rd International Conference on Software Engineering*, Atlanta, May 1978, pp. 230-242.
7. Pratt, T. W., “Pair Grammars, Graph Languages, and String-to-Graph Translations,” *Journal of Computer and System Sciences*, No. 5, 1971, pp. 560-595.
8. Boley, H., “Directed Recursive Labelnode Hypergraph: A New Representation-Language,” *Artificial Intelligence*, Vol. 9, 1977, pp. 49-85.
9. Stay, J. F., “HIPO and Integrated Program Design,” *IBM System Journal*, Vol. 15, No. 2, 1976, pp. 143-154.
10. Hartenstein, R. W., *Fundamentals of Structured Hardware Design*, North-Holland Publishing, Amsterdam, 1977.
11. Peterson, J. L. and Bredt, T. H., “A Comparison of Models of Parallel Computation,” *Information Processing 74*, 1974, pp. 466-470.
12. Scott, D., “Lattice Theory, Data Types and Semantics,” in *Formal Semantics of Programming Languages*, edited by R. Rustin, Prentice-Hall, New Jersey, 1972.
13. Kunii, T. L. and Kunii, H. S., “Architecture of a Virtual Graphic Database System for Interactive CAD,” *Computer-Aided Design*, Vol. 11, 1979, pp. 132-135.
14. Hansen, P. B., *The Architecture of Concurrent Programs*, Prentice-Hall, Englewood Cliffs, N.J., 1977.
15. Myers, G. J., “Reliable Software Through Composite Design,” Petrocelli Charter, N.Y., 1975.

APPENDIX 1

The predicates  $share(x,y)$ ,  $consist(x,y)$  and  $pass(x,y)$  can be formulated based on RGF as follows:

$$share(x,y) \triangleq \begin{cases} \text{TRUE ... if} \\ x = y \wedge (\exists d \in N)(brother(x,d) \wedge \\ channelaccess(x,d) \wedge brother(y,d) \wedge \\ channelaccess(y,d)) \\ \text{FALSE ... otherwise.} \end{cases}$$

$$consist(x,y) \triangleq \begin{cases} \text{TRUE ... if} \\ y \in sn(x) \wedge (\exists d \in N)(d \in chn(X) \wedge channel- \\ access(y,d)) \\ \text{FALSE ... otherwise.} \end{cases}$$

$$pass(x,y) \triangleq \begin{cases} \text{TRUE ... if } (\exists a \in A)(af(a) = (\{x\}, \{y\}) \\ \wedge atype(a) = \text{“u-link”}) \\ \wedge (\exists p \in pt(x))(styp(p) = \text{“parameter”}) \\ \text{FALSE ... otherwise.} \end{cases}$$

There,

$$chn(x) \triangleq sn(x) \cup pt(x),$$

$$parn(x) \triangleq \begin{cases} y \text{ ... if } (\exists y \in N)(x \in chn(y)) \\ \text{NIL ... otherwise (NIL stands for an} \\ \text{undefined value),} \end{cases}$$

$$brother(x,y) \triangleq \begin{cases} \text{TRUE ... if } parn(x) = parn(y) \\ \text{FALSE ... otherwise,} \end{cases}$$

$$styp(x) \triangleq \text{a value of the type field of system record } ns(x),$$

$atype(a) \triangleq$  a value of the type field of association record  $as(a)$ ,

$channelaccess(s,d) \triangleq$   $\left\{ \begin{array}{l} \text{TRUE ... if } stype(s) \in \\ \{ \text{"machine"}, \text{"routine"} \} \\ \wedge stype(d) \in \{ \text{"data} \\ \text{storage"}, \text{"parameter"}, \text{"access} \\ \text{channel"} \} \\ \wedge (\exists p \in pt(s), \exists a \in A) (stype(p) = \text{"ac-} \\ \text{cess channel"} \wedge (af(a) = (\{p\}, \{d\}) \\ \vee af(a) = (\{d\}, \{p\})) \\ \wedge atype(a) = \text{"d-flow"}) \\ \text{FALSE ... otherwise.} \end{array} \right.$

## APPENDIX 2

Survey operator  $SURV(R, NC, AC, SC)$  is defined as a sequence of R-operators in the following.

```

procedure SURV(R, NC, AC, SC)
/*comment
  NC  $\triangleq$  (NKEY1 of NFIELD1)  $\wedge$  ...  $\wedge$  (NKEYm of
    NFIELDm),
  AC  $\triangleq$  (AKEY1 of AFIELD1)  $\wedge$  ...  $\wedge$  (AKEYn of
    AFIELDn),
  SC  $\triangleq$  (struct by SKEY),
  X0, X1, ..., Xm, SX  $\triangleq$  a node set variable,

```

$A_0, A_1, \dots, A_n, SA \triangleq$  an arc set variable,  
comment end\*/

```

begin SURV
STEP1: Find all arcs {a} of design schema R, and set
  A0 := {a} and i := 0.
STEP2: Repeat [select from Ai all arcs {a} such that the
  arc record of arc a  $\in$  Ai has a value in a given search
  key AKEYi+1 at the field AFIELDi+1, set Ai+1 := {a}
  and increment i by 1] until i = n.
STEP3: If "ARC"  $\in$  SKEY then find all ancestor arcs
  {a} of arcs An in the arc hierarchy and set SA := {a}
  else SA :=  $\emptyset$ .
STEP4: Find all nodes {x} connected by arcs An  $\cup$  SA,
  set X0 := {x} and i := 0.
STEP5: Repeat [select from Xi all nodes {x} such that
  the node record of node x  $\in$  Xi has a value in a given
  search key NKEYi+1 at field NFIELDi+1, set Xi+1 :=
  {x} and increment i by 1] until i = m.
STEP6: If "NODE"  $\in$  SKEY then find all ancestor nodes
  {x} of the nodes Xm in the node hierarchy and SX :=
  {x} else SX :=  $\emptyset$ .
STEP7: Execute SEL(R, Xm  $\cup$  SX, An  $\cup$  SA).
end SURV

```

Note that for a given set  $S$ , if variable  $v$  is set variable  $v \in 2^S$ , while  $v \in S$  if  $v$  is a (usual) variable.

# An overview of a network design system

by W. E. BRACKER and B. R. KONSYNSKI

University of Arizona  
Tucson, Arizona

## INTRODUCTION

During the past few years there has been an increasing trend toward the development of on-line computer systems. By 1980 it is estimated that 70-80 percent of all larger computer installations will support some networking capability.<sup>1</sup> This trend has resulted in an increasing network user population and varying applications for computer-communication networks. Furthermore, due to economics, the prevalence of single-application networks is giving way to increasing numbers of multi-application networks.

This increase in the interconnection of computers has brought into focus the complexity of network design. While this is due partly to the size and diversity of computer networks, it is also due to the proliferation of available network hardware and facilities. As an example, there are over fifty different vendors (sales greater than one million/year) of data communication oriented hardware, and over twenty suppliers of data transmission facilities.<sup>2</sup>

## NETWORK DESIGN

As with any system, computer communications networks are made up of various interrelated components, all of which are critical to the network design process. Some of these components (i.e. multiplexors, modems, terminals, etc.) are physical in nature, that is, they specify a piece of hardware or software with certain performance properties. Some network components, however, are not physical in nature but rather are considered to be logical components of network design. These include such design inputs as response time, security levels, and specification of user interactions. The logical design components are as critical to the design process as the various physical elements. Figure 1 presents a partial list of the physical and logical design components.

The main objective function in designing a computer communications network is the production of a minimum cost network which satisfies user performance requirements and design criteria. It is usually the case that an exact quantification of these parameters and constraints is not always possible. This is due to the unavailability of exact data regarding various network components. Estimation of the design parameters and constraints is accomplished by a study of user needs, by various statistical methods or by adopting

industry standards. These approximations introduce errors in the design models employed during network design and analysis. The network planner also faces problems in deciding which network parameters are important to the design problem at hand; for example, some problems may require that modem turnaround time be specified exactly, while others may accept a rough approximation to this parameter.

### *Network design life cycle*

The network design life cycle shown in Figure 2 reflects the general system design life cycle which has been presented by various authors.<sup>3</sup> It should be stressed that the difficult phases of the evaluation of user needs and design parameter determination in the design life cycle are often slighted by network designers.

### **Organizational impact**

This design component is common to all systems design problems. We must completely analyze the impact of the proposed network or modifications to an existing network on the overall organizational structure. This impact analysis includes consideration of how much support upper management will give to the network during planning/design stages and eventual use of the system. In addition to analysis of the network impact on the current organizational structure, it is also necessary to measure the impact of the proposed system on future expansion and organizational goals.

### **Time-value of data**

Data sent to or requested from the network must be processed within a given time period; therefore, it is necessary to evaluate network requirements based on response or other time performance criteria. This function is difficult to measure because it requires estimates by users who often have little or no idea about what they really want. Time-value of data also refers to the fact that some data may be made available almost instantaneously over a network but may not be used immediately. An example case is where a monthly status report is generated within five minutes of a request but is not used for a week after production.

TOPOLOGY/NETWORK ARCHITECTURES  
 MULTIPLEXING/CONCENTRATION  
 LINE TYPES  
 TRANSMISSION TYPE AND METHODS  
 MODEM TYPES  
 ERROR DETECTION AND CORRECTION  
 MESSAGE FORMATS AND ROUTING  
 MESSAGE SWITCHING TECHNIQUES  
 TERMINAL TYPES  
 PROTOCOLS  
 BACKUP CONFIGURATIONS  
 INTERFACE STANDARDS  
 USER INTERACTIONS  
 CARRIER SELECTION

Figure 1—Design components and parameters.

### Traffic profiles

The definition of the information traffic profile is another area where quantification may not be possible; estimates of the type and arrival statistics of the incoming and outgoing data streams at a particular network node must be made. The network design process must provide some analysis and insight into the sensitivity of network performance due to variances of message structural estimates and assumptions.

### Reliability analysis

Reliability normally is analyzed based on equipment failure rates (MTBF, MTTR) and data transmission error rates. Equipment failure can be measured on a statistical basis or on past experience while message error rates are a function of the error detection and correction features of the network and the line transmission error properties.

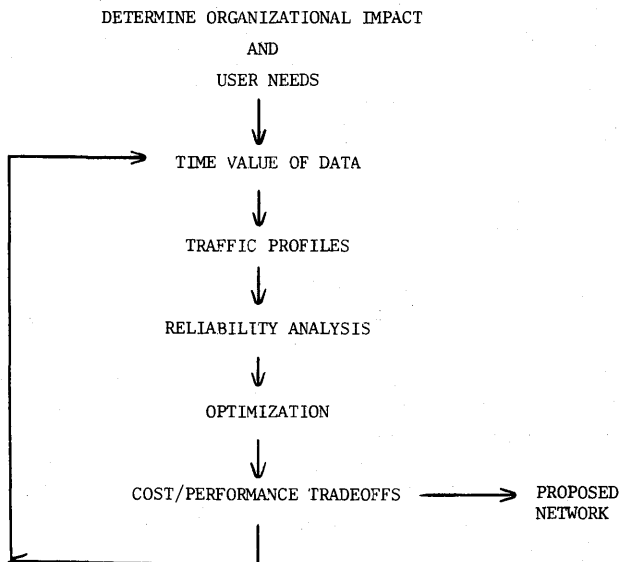


Figure 2—Network design life cycle.

### Design optimization

There are two major steps in the design optimization process: (1) performance evaluation based on a given set of parameters and (2) determination of cost/performance trade-offs. In the first step, a set of design parameters is specified and resultant performance is determined. The second step requires that a cost be determined for a given set of parameters and that cost/performance curves be established.

### NETWORK DESIGN SYSTEM (NDS)

The Network Design System developed at the University of Arizona is an attempt to formalize the network design life cycle into a computer-aided design process (see Figure 3). NDS uses a Decision Support Philosophy<sup>4</sup> which provides the network planner/designer with maximum flexibility in the creation of an optimized data communication network that meets previously discussed design criteria and constraints. Using this methodology, the network planner/designer is concerned about what NDS can do in terms of network design and not the details about how it goes about its processing tasks. In particular, interfaces to the various design models are made as user transparent as possible.

### Network planner/designer

The human interface to NDS is the network planner/designer; in most cases this consists of a group of individuals making up the planning/design team. These individuals state the network parameters and constraints to NDS using the

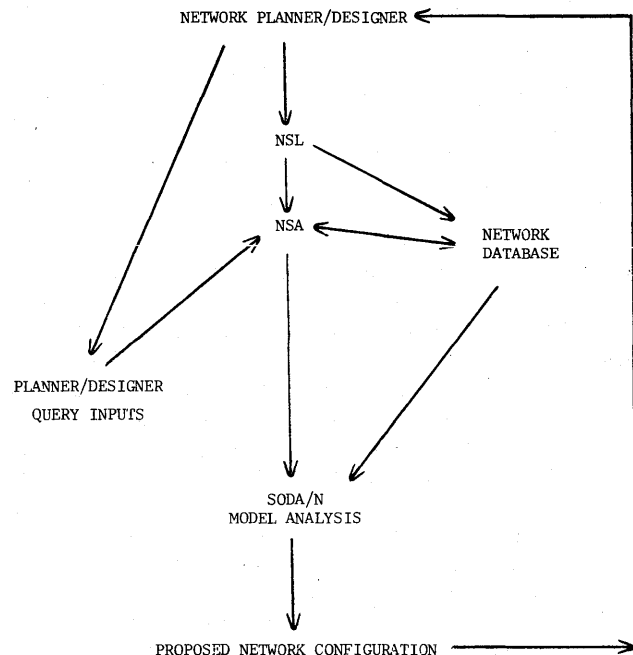


Figure 3—NDS Flow.

Network Specification Language (NSL) and the Network Statement Analyzer (NSA) query system.

*Network statement language (NSL)*

NSL is similar in form and structure to PSL<sup>5</sup> which is designed to provide a user with methodology for the statement of requirements of an information processing system. NSL, in a form compatible with PSL, allows a user to state design requirements for computer communication networks. The language provides an interface with a set of design models and is the main user contact with NDS. The current implementation of NSL has fourteen sections and thirty-five connector words or individual statements acting as "adjectives" in describing their particular sections. NSL allows the description of the physical and logical network constructs discussed previously. NSL section types, individual statements, and connector words are shown in Table I.

NSL is specified using the META/Generalized Analyzer<sup>6</sup> methodology of the ISDOS project based at the University of Michigan. Both META and the Generalized Analyzer are in a significant prototype stage and are not currently available to the general public.

META analyzes the description of NSL and produces a database containing the language structure. After NSL has been specified and processed by META, the META Generalized Analyzer (GA) processes user-supplied NSL problem statements, analyzes the syntax and portions of the semantic relations and iteratively builds the Network Database. NSA uses the constructed database for its processing requirements.

Basic flow between NSL, META and the Generalized

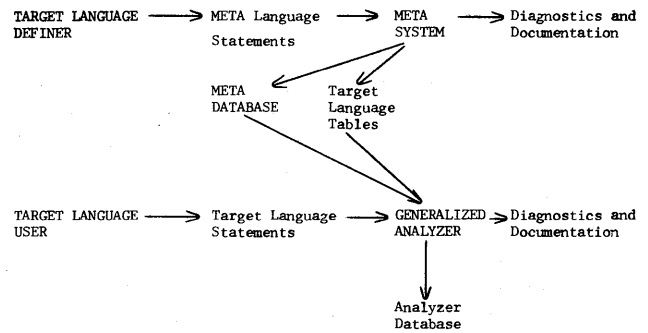


Figure 4—META system and generalized analyzer.

Analyzer is shown in Figure 4. The target language (NSL) is defined using META constructs; META then analyzes the target language and creates a database containing all target language objects types and their language inter-relationships. At this stage, META produces error diagnostics and various reports for further analysis and documentation.

After the target language has been successfully processed by META, sets of tables for the Generalized Analyzer are produced. The Generalized Analyzer enters the NSL network description into the network database and produces a series of reports which can be used by the network designer for documentation and/or analysis purposes.

*Network statement analyzer*

The Network Statement Analyzer is the NDS processor designed to accept network specifications produced by NSL and also obtain and process design inputs and report requests obtained by the query system. The NSA query processor produces a menu which allows the NDS user to select the supported NSA statement types as shown in Table II.

Consistency checks on the database produced by META/GA are performed by NSA. An NDS user can select checks based on various hardware connections: among these, whether all nodes in a proposed network can be reached by all other nodes, making sure that line and terminal speeds match, and verification of proper multiplexor/concentrator connections.

*SODA/N*

The Network System Optimization and Design Algorithms provide a set of models used to evaluate various design al-

TABLE I.—NSL Reserved Words

NSL SECTION TYPES

NODE	HOST
TOPOLOGY	DATABASE
TRAFFIC	MULTIPLEXOR
INFORMATION-USER	TERMINAL
TERMINAL-USER	INFORMATION-CENTER
APPLICATION	LINE
REPORT	DATA-SET

NSL STATEMENT TYPES

TERMINAL	INTELLIGENCE	HOST
MULTIPLEXOR	INFORMATION-USER	RESPONSE
TOPOLOGY	TERMINAL-USER	SIZE
HOST	SEE-MEMO	DATABASE
LINK	TRAFFIC	SEE-MEMO
FAN-IN	FLOW	HAPPENS
FAN-OUT	KEYING	LOCATION
PROTOCOL	PRIORITY	RELIABILITY
TYPE	GENERATES	COST
MODE	RECEIVES	SECURITY
ERROR	SYNONYMS	DESCRIPTION
KEYWORDS	APPLICATION	

NSL CONNECTOR WORDS

TO	BY
FOR	VIA
IS	ARE

TABLE II.—NSA Supported Statements

NDS/NSA STATEMENT TYPE	PURPOSE
QUERY	Activate NSA query processor
MODEL	Activate NDS supported models
CONSISTENCY	Perform consistency checks on the network database
MACRO	Activate the NSL macro pre-processor
INVENTORY	Various network reports



ternatives. In the current implementation of NDS, five design models are available; overall model implementation and integration philosophy is to create a design interface which requires minimum user interaction. Outputs from the models are also processed by NDS so that minimum user interpretation is needed; the user is only concerned about what information the models provide and not details on how they provide that information.

The models derive their inputs from two sources: NSL design specifications and NSA user interactions. The NSL design specifications produce a set of initial network conditions and assumptions to the design problem, while user interaction with NSA produces various design constraints and performance criteria. As an example, Table III shows the interaction between NSL, NSA and a capacity assignment model.

### EXAMPLE

To illustrate the various components, the NSL section types NODE, TOPOLOGY, and TRAFFIC are implemented using the META methodology. The syntax description of these sections and associated statements are shown in Appendix A.

After the NSL syntax for the three sections and associated statements has been defined, the META representation for the abbreviated NSL is produced (Appendix B); this representation requires that all keywords, noise words and object types be defined. In addition, the relationship between objects must be specified along with template forms of the statements themselves. As was discussed previously, the result of this process is a database containing all the NSL language relationships and interface tables for the Generalized Analyzer. The system also produces a set of reports showing language structure and interrelations; a sample of this report type is shown in Appendix C.

NSL statements describing the proposed network design are input to the Generalized Analyzer. The Analyzer checks the incoming NSL syntax and places the NSL constructs

into the META database. After all NSL descriptions have been processed, the Network Database is ready for access by NSA and its associated reports and models.

In order to show this process in greater detail, consider the small network shown in Figure 5 consisting of four NODEs (NODE1-NODE4) and five communication links (LINE1-LINE5). Traffic rates between any two node pairs are assumed to be symmetric and statistically independent of traffic between other node pairs. The NSL description of the simple example is shown in Appendix D.

Network topology showing direct connections between nodes is described in the TOPOLOGY-SECTION, while description of the individual nodes are shown in the NODE-SECTIONS. It is assumed that each node has two terminals, one multiplexor, and a host. Notice that the NODE-SECTIONS describe the hardware available at each node along with that node's location using a V/H coordinate scheme (LOCATION Statement).

Once the NSL description has been processed by the Generalized Analyzer and input to the Network Database, NSA is activated to produce database reports or activate various design/analysis models. In the example, we consider activation of a capacity assignment model.<sup>7</sup> This model establishes optimal link capacities based on network topology, message routing (assumed to be shortest path) and message traffic profiles. In the current implementation, Poisson Message arrival rates, exponential node service with infinite buffering are assumed. NSA accesses the Network Database and queries the user in order to establish the model input parameters. An example run of this model is shown in Appendix E assuming the Appendix D NSL description.

### SUMMARY

The Network Design System provides an easy to use network planning and design tool; in addition, it allows a methodology of describing and evaluating existing networks. NSL statements are analyzed by the Network Statement Analyzer which, in turn, provides a Network Database, consistency checking, report generation, and model interfaces. Using the NDS approach, both existing and proposed systems are thoroughly analyzed. In addition, the top-down approach which is used with NDS allows the system planners/designers to maintain a perspective of the overall design

TABLE III.—Capacity Model Parameters

#### NSL-SUPPLIED PARAMETERS

	Parameter	Obtained From
$T_{jk}$	Messages/sec between two network nodes $j, k$	TRAFFIC-SECTION
$N_{jk}$	Connection matrix showing direct connections between node $j$ and node $k$	TOPOLOGY-SECTION

#### NSA-QUERY PARAMETERS

$U_i$	Average length of messages over communication link $i$
$C$	Overall network capacity
$R$	Message Routing

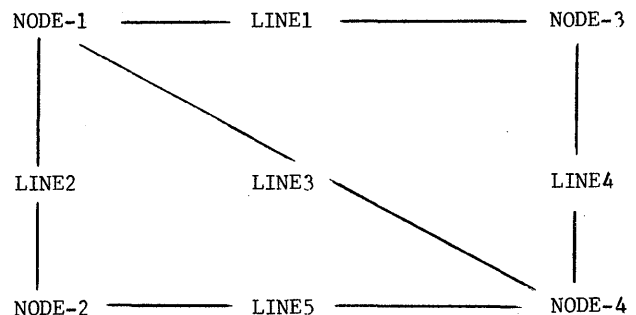


Figure 5—Sample network.

goals while at the same time allowing access to desired levels of detail in the design process.

## REFERENCES

1. McLaughlin, R. A., "Piecing Together The Datacom Industry," *Data-mation*, July, 1979, pps. 110-137.
2. Bracker, W. E., "On the Automated Design of Data Communication Networks," Working Paper, Univ. of Arizona, Dept. of MIS, 1979, unpublished.
3. Cougar, D., "Evolution of Business System Analysis Techniques," *Computing Surveys*, September 1973, reprinted in Cougar and Knapp, *System Analysis Techniques*, Wiley, 1974.
4. Konsynski, B. and Bracker, W., Computer-Aided System Design Tools Integration," MIS Technical Report, University of Arizona, 1979.
5. ISDOS Project, "Problem Statement Language (PSL): Introduction and User's Manual," ISDOS Project Report, University of Michigan, 1979.
6. Yamamoto, Y., "ISDOS META System Memorandum META-1,2," ISDOS Project, Dept. of Industrial and Operations Engineering, The University of Michigan, 1978.
7. Schwartz, M., *Computer-Communication Network Design and Analysis*, Ch. 4, Prentice Hall, Englewood Cliffs, N.J., 1977.
8. Konsynski, B., "A Model of Computer-Aided Definition and Analysis of Information System Requirements," Ph.D. Dissertation, Purdue University, 1976.

## APPENDIX A: SAMPLE NSL SYNTAX

```
##### NODE-SECTION #####
NODE-SECTION name(s);
SYNONYMS ARE synonym-name(s);
DESCRIPTION;
comment-entry;
KEYWORDS ARE keyword-name(s);
SEE-MEMO memo-name(s);
TERMINALS ARE terminal-name(s);
MULTIPLEXORS ARE multiplexor-name(s);
HOSTS ARE host-name(s);
LOCATION IS system-parameter, system-parameter;
##### TOPOLOGY-SECTION #####
TOPOLOGY-SECTION name(s);
SYNONYMS ARE synonym-name(s);
DESCRIPTION;
comment-entry;
KEYWORDS ARE keyword-name(s);
SEE-MEMO memo-name(s);
```

```
LINK {node-name ! terminal-name ! multiplexor-name}
TO
{node-name ! terminal-name ! multiplexor-name}
VIA line-name;
```

```
##### TRAFFIC-SECTION #####
```

```
TRAFFIC-SECTION name(s);
```

```
SYNONYMS ARE synonym-name(s);
```

```
DESCRIPTION;
comment-entry;
```

```
KEYWORDS ARE keyword-name(s);
```

```
SEE-MEMO memo-name(s);
```

```
TRAFFIC node-name TO node-name FLOW system-
parameter;
```

## APPENDIX B: SAMPLE NSL META REPRESENTATION

```
KEYWORD LOCATION;
SYNONYMS LOC;
```

```
KEYWORD TRAFFIC;
SYNONYMS TRAF;
```

```
KEYWORD FLOW;
SYNONYMS FL;
```

```
KEYWORD LINK;
SYNONYMS LI;
```

```
NOISE-WORD ARE;
NOISE-WORD IS;
NOISE-WORD PER;
NOISE-WORD TO;
NOISE-WORD VIA;
```

```
OBJECT NODE-SECTION;
SYNONYMS N-S, NS;
NMCODE NMNODE 1;
```

```
OBJECT TRAFFIC-SECTION;
SYNONYMS T-S, TS;
NMCODE NMTRAF 2;
```

```
OBJECT TOPOLOGY-SECTION;
SYNONYMS TOP-S, TOPS;
NMCODE NMTOPO 3;
```

```
PROPERTY INTEGER-VALUE;
APPLIES ALL;
VALUES INTEGER;
```

PROPERTY STRING-VALUE;  
 APPLIES ALL;  
 VALUES ANY-VALUE;

PROPERTY NUMBER-VALUE;  
 APPLIES ALL;  
 VALUES ANY-VALUE;

RELATION LOCATION-RELATION;  
 PARTS LOCATION-OBJECT-PART, LOCATION-  
 PART-1, LOCATION-PART-2;

COMBINATION LOCATION-OBJECT-PART NODE-  
 SECTION WITH LOCATION-PART-1 VALUE-FOR  
 INTEGER-VALUE WITH LOCATION-PART-2  
 VALUE-FOR INTEGER-VALUE;

CONNECTIVITY MANY LOCATION-OBJECT-PART  
 ONE LOCATION-PART-1, LOCATION-PART-2;

CONNECTION-TYPE T5;  
 RTCODE RTLOCA 50;  
 STORED LOCATION-OBJECT-PART 3, LOCATION-  
 PART-1 1, LOCATION-PART-2, 2;

STATEMENT LOCATION-STATEMENT;  
 USED LOCATION-OBJECT-PART LOCATION-  
 RELATION;  
 FORM LOCATION IS LOCATION-PART-1,  
 LOCATION-PART-2;

RELATION TRAFFIC-RELATION;  
 PARTS TRAFFIC-OBJECT-PART, TRAFFIC-PART-1,  
 TRAFFIC-PART-2, TRAFFIC-PART-3;  
 COMBINATION TRAFFIC-OBJECT-PART TRAFFIC-  
 SECTION WITH TRAFFIC-PART-1 NODE-SECTION  
 WITH TRAFFIC-PART-2 NODE-SECTION WITH  
 TRAFFIC-PART-3 VALUE-FOR NUMBER-VALUE;

CONNECTIVITY ONE TRAFFIC-OBJECT-PART  
 MANY TRAFFIC-PART-1, TRAFFIC-PART-2,  
 TRAFFIC-PART-3;

CONNECTION-TYPE F4;  
 RTCODE RTTRAA 55;  
 STORED TRAFFIC-OBJECT-PART 1, TRAFFIC-PART-  
 1 2, TRAFFIC-PART-2 3, TRAFFIC-PART-3 4;

STATEMENT TRAFFIC-STATEMENT;  
 USED TRAFFIC-OBJECT-PART TRAFFIC-  
 RELATION;  
 FORM TRAFFIC TRAFFIC-PART-1 TO TRAFFIC-  
 PART-2 IS TRAFFIC-PART-3;

RELATION TOPOLOGY-RELATION;  
 PARTS TOPOLOGY-OBJECT-PART, TOPOLOGY-  
 PART-1, TOPOLOGY-PART-2, TOPOLOGY-PART-  
 COMBINATION TOPOLOGY-OBJECT-PART  
 TOPOLOGY-SECTION WITH TOPOLOGY-PART-1

TOPOLOGY-SECTION WITH TOPOLOGY-PART-2  
 TOPOLOGY-SECTION WITH TOPOLOGY-PART-3  
 VALUE-FOR STRING-VALUE;

CONNECTIVITY ONE TOPOLOGY-OBJECT-PART  
 MANY TOPOLOGY-PART-1, TOPOLOGY-PART-2,  
 TOPOLOGY-PART-3;

CONNECTION-TYPE F4;  
 RTCODE RTTOPA 60;  
 STORED TOPOLOGY-OBJECT-PART 1, TOPOLOGY-  
 PART-1 2, TOPOLOGY-PART-2 3, TOPOLOGY-PART-  
 3 4;

STATEMENT TOPOLOGY-STATEMENT;  
 USED TOPOLOGY-OBJECT-PART TOPOLOGY-  
 RELATION;  
 FORM LINK TOPOLOGY-PART-1 TO TOPOLOGY-  
 PART-2 VIA TOPOLOGY-PART-3;

#### APPENDIX C: META SAMPLE REPORT-OBJECT SUMMARIES

Object name = NODE-SECTION Synonym(s) = N-S, NS

Relation name = LOCATION-RELATION  
 Part name = LOCATION-OBJECT-PART  
 Statement name = LOCATION-STATEMENT  
 Form = 1: LOCATION IS LOCATION-PART-1,  
 LOCATION-PART-2 ;

Relation name = TRAFFIC-RELATION  
 Part name = TRAFFIC-PART-2  
 \*\*\* No usages

Relation name = TRAFFIC-RELATION  
 Part name = TRAFFIC-PART-1  
 \*\*\* No usages

Object name = TOPOLOGY-SECTION  
 Synonym(s) = TOP-S, TOPS

Relation name = TOPOLOGY-RELATION  
 Part name = TOPOLOGY-OBJECT-PART  
 Statement name = TOPOLOGY-STATEMENT  
 Form = 1: LINK TOPOLOGY-PART-1 TO  
 TOPOLOGY-PART-2 VIA TOPOLOGY-PART-3 ;

Relation name = TOPOLOGY-RELATION  
 Part name = TOPOLOGY-PART-2  
 \*\*\* No usages

Relation name = TOPOLOGY-RELATION  
 Part name = TOPOLOGY-PART-1  
 \*\*\* No usages

Object name = TRAFFIC-SECTION Synonym(s) = T-S, TS

Relation name = TRAFFIC-RELATION  
 Part name = TRAFFIC-OBJECT-PART  
 Statement name = TRAFFIC-STATEMENT  
 Form = 1: TRAFFIC TRAFFIC-PART-1 TO  
 TRAFFIC-PART-2 IS TRAFFIC-PART-3 ;

APPENDIX D: NSL OF EXAMPLE NETWORK

TOPOLOGY-SECTION TOP 1;

LINK NODE-1 TO NODE-3 VIA LINE1;  
 LINK NODE-1 TO NODE-2 VIA LINE2;  
 LINK NODE-1 TO NODE-4 VIA LINE3;  
 LINK NODE-2 TO NODE-3 VIA LINE4;  
 LINK NODE-2 TO NODE-4 VIA LINE5;

TRAFFIC-SECTION TRAFFIC1;

LINK NODE-1 TO NODE-2 FLOW 9.05;  
 LINK NODE-1 TO NODE-3 FLOW 6.12;  
 LINK NODE-1 TO NODE-4 FLOW 3.00;  
 LINK NODE-2 TO NODE-3 FLOW 4.50;  
 LINK NODE-2 TO NODE-4 FLOW 1.00;  
 LINK NODE-3 TO NODE-4 FLOW 10.8;

NODE-SECTION NODE-1;  
 TERMINALS ARE T1-1, T2-1;  
 MULTIPLEXOR IS MUX-1;  
 HOST IS CPU1;  
 LOCATION IS 10, 25;

(REPEAT FOR NODE-2 ... NODE-4)

APPENDIX E: NDS EXAMPLE

NDS—UNIVERSITY OF ARIZONA VERSION 1.0 8/3/  
 79 10:20

→NSL = TEST. NSL (File created in Appendix D)  
 →LISTING→TEST.LST (Source/Diagnostics File)

→DATABASE→TEST.DB (META Produced Database)

\*\* NO DIAGNOSTICS \*\*  
 SECTIONS PROCESSED:6  
 STATEMENTS PROCESSED:55  
 OUTPUT FILE AND DATABASE FILE WRITTEN

\*\* NDS/NSL COMPLETE 8/3/79 \*\*

\*\* NDS/NSA VERSION 1.0 8/3/79 \*\*

OPTIONS:

1. EXIT
2. MODELS
3. REPORTS

OPTION→2

\*\* NSA MODEL ANALYSIS \*\*

ACTIVE MODELS:

1. CONCENTRATOR LOCATOR
2. CAPACITY ASSIGNMENT
3. TERMINAL LOCATOR
4. SPANNING TREE

MODEL→2

\*\* CAPACITY ASSIGNMENT \*\*

REQUIRED SECTIONS:

TOPOLOGY-SECTION

TRAFFIC-SECTION

\*\* ALL REQUIRED SECTIONS CONSISTENT \*\*

ROUTING→SHORT

CAPACITY→1000

\*\* MODEL COMPLETE:CAPACITY \*\*

RESULT FILE→TTY

MODEL:CAPACITY

ROUTING:SHORT

CAPACITY:1000

LINK	CAPACITY(BPS)
LINE1	200
LINE2	450
LINE3	110
LINE4	100
LINE5	120

LINE1 200

LINE2 450

LINE3 110

LINE4 100

LINE5 120

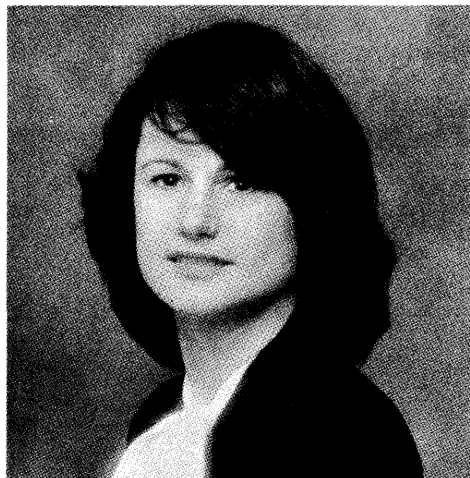
\*\* ALL CONSTRAINTS MET

\*\* NDS/NSA VERSION 1.0 TERMINATED 8/3/79 \*\*



## Computers and Entertainment

In response to a paper I wrote last year on computer technology and the movie industry<sup>1</sup>, Dick Thompson, a special effects professional, commented, "There is much more involvement in computer technology in "show business" than anyone suspects. You are just beginning to turn over the rock—or is it a can of worms?" Dick's question remains to be answered because the relationship between the entertainment industry and the computer industry is yet in its infancy. I think entertainment is truly the last (earthly) frontier for automation and over the next decade we will see exciting results of this union. Indications of the importance of computers in entertainment are already surfacing: The Society of Motion Picture and Television Engineers devoted an entire day to computer applications at its 1979 Conference; several computer-based systems have been nominated for Technical Achievement Academy Awards in 1980. The entertainment industry is thus beginning to acknowledge the significant role of the computer. Appropriately, the computer industry also recognizes its current and potential contributions to entertainment through six technical paper and panel sessions at the 1980 NCC. The sessions will provide an overview of computer usage in movies and television, the performing arts, amusements, and home entertainment/services.



Suzanne Landa  
*Area Director*

Three of the sessions focus on computer applications in movie and television production. The first explores computer usage for special audio and visual effects. Deitrick describes an automated computer-controlled editing sound system (ACCESS) that provides the editor with the capability to digitally create, modify and store sound instantaneously. ACCESS, an Academy nominee this year, has been used in the production of numerous movies and television shows, increasing output fivefold. Another Academy contender is the Automatic Camera Effects System (ACES) described by Crane and Snyder. By controlling camera and model movements, ACES achieves a level of accuracy, repeatability, and originality for 3-dimensional special effects not possible with manual methods. In King's paper on the Emmy award winning system MAGICAM, the use of a computer to maintain precise perspective in matting scenes of normal and miniature sized objects is illustrated.

A panel session on computer applications in film animation will cover computer systems allowing an animator to enter key sketches at a terminal with the computer extrapolating the frames between these sketches. Color, shadow and shading are then added through software options. Finally, the images are automatically filmed off a high-resolution CRT. Another approach to be demonstrated is to totally synthesize images within the computer.

The third movie and television session will focus on computer support behind the scenes. A panel will review such applications areas as casting, budgeting, and on-location cost control and will also address employment opportunities in the movies for the data processing professional. A working model of an automated sound stage will be demonstrated.

Along with movies and television, the performing arts are beginning to utilize computer technology. In Smoliar's paper on dance, computer graphic aids in choreography are explored. For drama, a guest speaker will survey data processing applications in theatrical administration and operations. Another speaker will focus on computer music.

Computer technology is already an integral part of most amusements today. Parks, casinos, planetariums, and sporting events are capitalizing on mini- and micro-computer technology. Stover and Snyder's paper describes a mini-computer system used in designing audio-animatronic shows at Disneyland. Eifler's paper surveys the already pervasive use of computers in scoring sports events.

<sup>1</sup> Landa, Suzanne, "Computer Technology and the Movie Industry," *Proceedings of the 1979 National Computer Conference*, AFIPS Press, June 1979, p.1+.



# A minicomputer system for audio-animatronics show data generation

by PHILIP C. STOVER and R. DAVID SNYDER

Walt Disney Productions  
Burbank, California

## INTRODUCTION

Audio-Animatronics® shows have been produced by Disney since the introduction of several attractions at the 1964 New York World's Fair.\* Since then a variety of shows have been permanently installed at both Disneyland and Walt Disney World. They typically consist of a stage, or some enclosed show area, and a variety of special lighting effects, mechanized characters and other movable stage equipment. Some of the shows include: "Great Moments with Mr. Lincoln," "The Country Bear Jamboree," and "Pirates of the Caribbean."

The technology used to produce these shows has been constantly evolving and improving toward a more complex and versatile medium for the animator's use. A significant advance in this technology was the move to all digital show data that occurred in 1968-1969. Since then a minicomputer has been an integral part of the show development system. It allows the animator to easily generate, review and edit this digital database to produce the animated show. The name given to this system was DACS (Digital Animation Control System).

In 1971, the minicomputer used for DACS was a Honeywell 516. It has served this purpose at both Disneyland and Walt Disney World for every Audi-Animatronics show that has been produced since then. In 1979, however, with the upcoming expansion at Walt Disney World and the Tokyo Disneyland project, it became necessary to upgrade DACS to a more modern computer system. This second generation DACS is the subject of this paper.

## THE DIGITAL AUDIO-ANIMATRONICS SHOW

All components of the digital Audio-Animatronics shows are controlled by a single, unified channel-addressing scheme. This design allows a single show to include up to 1000 separate channels. Each channel can be an eight-bit analog value or eight separate digital subchannels. Analog channels are

used for the bulk of the show to provide signals for most of the figure movements, the light dimming functions, and other smooth linear motions. The digital subchannels are used for on/off functions such as eye blinks, platform lifts, and other special effects.

Figure 1 shows the show data transmission system. All of the show data are de-multiplexed from a cabinet called an RTU (Remote Terminal Unit). From this point, signals are routed through servo control cabinets (in the case of the air- and oil-actuated figure movements) or directly to other control points. The RTU cabinet is located immediately adjacent to the show area and receives data over twisted pair conductors from a remote central control area. All data are transmitted by serial synchronous communication over these cables. The show data reside on a fixed head disk and are played back by a hardwired controller. This combination is called a Show Control Unit.

The Show Control Unit in turn is synchronized using a telemetry encoding scheme to a multi-track audio tape machine which provides all of the show's audio. In this way the show actions, including mouth movements, are always in complete sync with the audio soundtrack, regardless of tape speed variations. The data update rate is the same as the movie industry's 24 frames per second. Data that are not changing are updated less frequently to save disk space.

Future Show Control Units will likely incorporate another medium for show data storage such as bubble memory. The show data generation process, however, will continue to be identical regardless of the playback medium used.

## Show data generation

During show data generation, a minicomputer is substituted for the Show Control Unit to provide data transmission to the show being animated. In addition, a special purpose control panel called an Animator's Console is connected to the minicomputer to provide the input device for the animator to use to generate show data. Figure 2 illustrates this connection, and shows the Animator's Console located in front of the stage area to provide a clear view of the show for the animator.

By manipulating knobs and switches on the Animator's

\* Finch, Christopher, *The Art of Walt Disney*, (New York, Harry N. Abrams, Inc., 1975), p. 152.



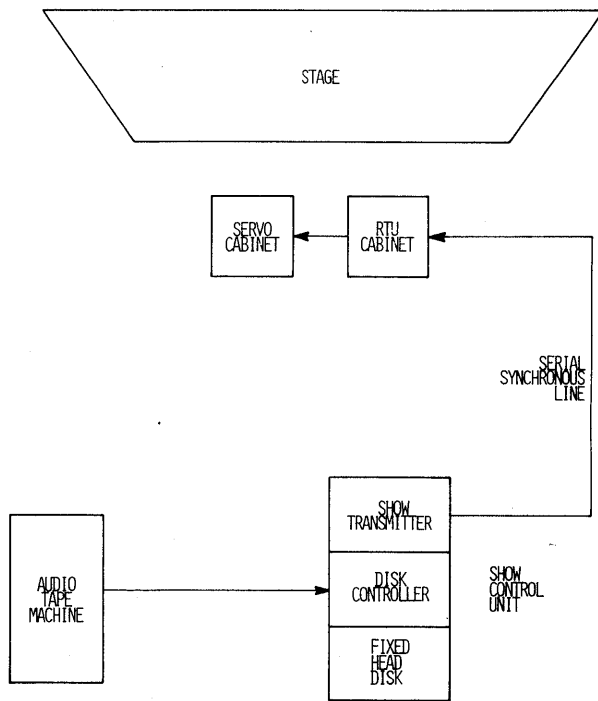


Figure 1—Normal animated show playback configuration

Console, a show database is gradually built up, a few channels at a time. The DACS minicomputer provides immediate visual feedback via the RTU and includes various editing and playback features. During animation sessions, data are stored on the minicomputer's disk drives. After animation is complete, show data are transferred to the Show Control Unit disks.

### The computer system

A Data General Eclipse S250 forms the basis of the second generation DACS. The show data transmission and channel addressing design was left identical to the original DACS so that the new system could be used to reanimate existing shows. A redesigned Animator's Console and single custom interface board for the S250 were mated to standard Data General components. The Eclipse system includes three 50 megabyte disk drives. One is used for system support and the remaining two provide redundant database storage during show data generation.

The Animator's Console was redesigned to incorporate more functions in a much smaller package. It includes a large number of switches, knobs and displays to provide a variety of data manipulation functions for the animator. The microprocessor located in the Animator's Console scans the switches and knobs and drives the console indicators. Message blocks are sent continuously at 30 hertz to the Eclipse incorporating the current analog pot positions as well as the switch scanning information. The microprocessor receives message blocks from the Eclipse with indicator and display

information. Error checking is incorporated at both ends of the communication link. The Eclipse is responsible for interpreting all of the console requests and sending back the proper console response along with show data to the RTU. With the console the animator may position himself anywhere within the entire show and restrict his area of operation by use of scene limits. He may further restrict the working space to a subset of all the channels used for the show to focus his attention on a single figure or group of actions. With the console he may play back the existing show data forward or reverse at rates between one and 24 frames per second. He may, in addition, repeat the current scene continuously for critical viewing. New data may be deposited on the disk and viewed immediately by using any or all of the 32 pots and 16 switches on the console. The pots may be dynamically assigned to any of the analog show channels and the switches likewise may be assigned to any of the digital subchannels. Following this assignment new data may be entered or old data changed one frame at a time or continuously at rates between one and 24 frames per second.

The second generation DACS system incorporates many features not available at the time the first generation system was designed. An extremely straightforward hardware and software interface to the custom components of this system has been chosen. Standard RDOS operating system features were used wherever possible, and 95 percent of the code for this system has been generated in the FORTRAN V language. The Honeywell 516 DACS was coded entirely in assembly language without benefit of a true operating system.

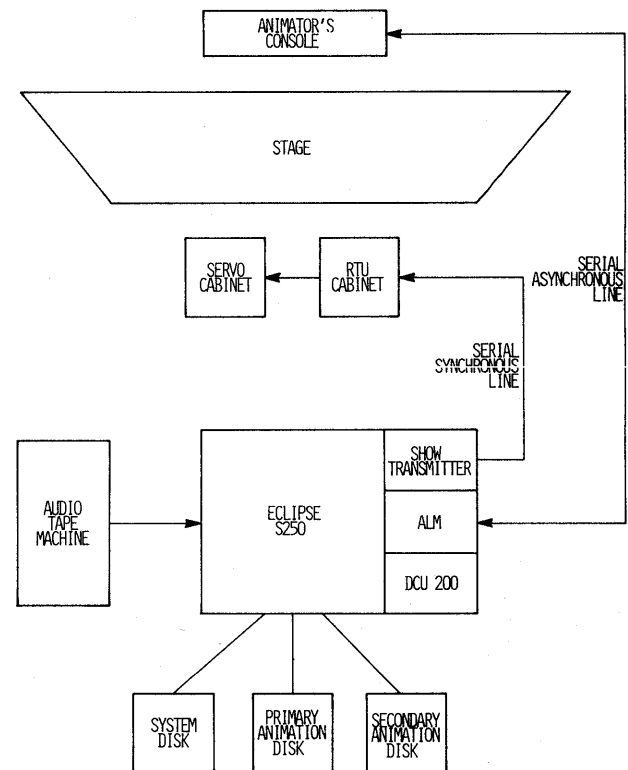


Figure 2—Animation data generation configuration

The communication link to the Animator's Console was accomplished with a standard Asynchronous Line Multiplexor operating at 19.2K baud. This full duplex line uses two twisted pairs with outboard line drivers and receivers to operate at distances up to one mile. Character interrupts are buffered at the Eclipse by a DCU 200 programmable I/O processor. The DCU 200 incorporates message checking and shields the S250 processor from the character interrupt processing overhead. Data are transferred via DMA channel directly from main processor memory to DCU 200 processor memory and vice versa. A single custom I/O board is inserted in the S250 chassis. It includes a DMA interface for show data transmission, a 24 hertz clock for show timing, and a Time Code Translator interface for audio sync. This board is the only non-standard connection needed to the S250 for this application.

The software design relies heavily on the RDOS multi-tasking operating system features. Separate tasks are used to partition necessary functions into multiple asynchronous processes, leaving a minimum of interrupt driven code. This partitioning results in an extremely flexible and easy system to maintain and enhance. The FORTRAN V language was found to be fast enough to perform all real-time data manipulation and computations required to keep up with the 24 frames per second rate. Disk data access is accomplished using standard direct block I/O and contiguous files. Real-

time data response is insured by multiple block read and write buffers. The buffer size was chosen to overcome worst case disk rotational latency and still provide up to 24 frames per second throughput. All console button and indicator lamp assignments are implemented in software, making changes in console function and button sequences easy.

The DACS minicomputer also provides all of the data manipulation and housekeeping functions performed off-line before and after animation sessions. One of the most important of these functions is data compression. In this operation up to four 33 megabyte files are compressed from tape to a single 2 megabyte disk file. This compressed file is then transferred to the Show Control Unit for repetitive playback. Compression is achieved by eliminating redundant data entries for successive frames, providing only occasional updates to correct any possible noise errors.

## CONCLUSION

A second generation system for Audio-Animatronics show data generation has been developed taking advantage of standard hardware and software products available today. This new system enhances the capability of the animator to produce more complex shows with higher quality in a shorter time frame.



# Computers and sports: a natural marriage

by THOMAS A. EIFLER

Honeywell  
Waltham, Massachusetts

## INTRODUCTION

The use of computers to score and time the outcome of various athletic events is almost as fast-growing as several of the sports themselves. From Austria to Atlanta, from Switzerland to Cleveland, in bowling alleys, on basketball courts, and at the finish lines of long distance foot races, computers are rapidly becoming as much a part of the world of athletic competition as the joy of victory and the agony of defeat. Whether serving as surrogate officials, omniscient scorekeepers, or stationary superscouts, these perfectly objective machines are adding to the enjoyment of both spectator and participant alike.

In this age of instant replay and on-the-spot analysis by TV commentators, there is no margin for error. Final results are expected—or in some cases, required—immediately upon conclusion of a particular event. The human mind is simply unable to tell who finishes 6,142nd in a race of 7,000, let alone how far ahead of the next runner this individual is. A computer can. The shrewdest mathematician cannot instantaneously calculate the average score of several judges at a gymnastics competition. A computer can.

Here, then, is a brief look at some of the fascinating ways in which computers are being used to monitor results of athletic competition:

### *Long distance running*

For the past five years, computers have been used to score and time the Boston Marathon. The system is fairly sophisticated, but in layman's terms it works this way: before the runners start crossing the finish line in droves (i.e., for about the first 1,000 finishers in a 7,000 field race), a button is pushed each time a participant completes the race. This act notifies the computer that "a body" has crossed the line. At this point, the system doesn't know (or care) whether this body is male or female, young or old, official or unofficial; all it knows is that a runner has completed the race at a specific time.

The runners then line up in as many as eight chutes, each of which can be 100 or more yards long and each of which can hold 300-400 runners. At the end of the chutes, officials record each runner's number, in order. If an unofficial entrant appears at the end of the line, this fact is noted and the runner's name and order of finish do not appear in the final

listing that shows each participant's statistics. In other words, if the printout generated at the end of the race indicates that a runner finished 1,009th, this means that he or she completed the race after 1,008 other *official* entrants.

Meanwhile, back at the finish line, runners eventually begin to cross the line in bunches—30, 40, 50 at a time. Since the button just cannot be pushed fast enough, a different system is then put into effect. A particular runner is "spotted" on his way to the finish line and, as he crosses, his time and number are recorded. Fifteen seconds later, another runner is spotted in the same way. The system then knows that all runners who cross the line between runner A and runner B do so in the 15-second interval between runner A's time and runner B's.

If 600 runners cross in that 15 seconds, the system distributes this information over the allotted time. Thus, a runner's time could be off by a second or two, but once 1,000 runners have already crossed the line, this is not really all that important. What counts most and what the runners are interested in is how they do against their peers. This the system tells them very quickly and very accurately.

Once all the runners' numbers and times have been entered, the computer matches them up and creates a printout. Thus, within minutes after the 2,000th runner crosses the line, he'll know his exact time and order of finish. The same is true for the 4,000th, 5,000th—even the 7,000th runner.

At last year's Bonne Bell Championship for Women (5,035 runners), Honeywell computers were used in conjunction with bar code readers for the first time. This eliminated the time-consuming recording of numbers at the end of the chutes. Instead, the runners merely tore off their bar codes, which were read instantly, thereby enabling Honeywell to produce printouts in seconds rather than in minutes.

Computers have been scoring and timing races around the country for the past few years, including the Gasparilla Long Distance Classic in Tampa, Fla; the Chicago Long Distance Classic; the Cleveland Heart Run; and the Purity Supreme Heartbreak Hill Road Race in Boston.

### *Track and field*

In the spring of 1979, the editors of *Runner's World* magazine approached members of Honeywell's Public Relations department and asked if the company would be interested in scoring and timing the publication's second annual Cor-

porate Cup competition. This series of events consists of "track meets" at which numerous U.S. companies are pitted against each other in eight different races. Some are open to all employees, some to women only, some to runners over 40, etc. Last year, meets were held in eight different cities on seven consecutive week-ends, with the finals staged in San Francisco.

Less than 10 days after the editors made their request, a software program capable of processing individual runner times, team performance, and overall standings was up and running. Honeywell employees, portable terminals in hand, attended each of the eight meets and entered the appropriate data after each race, usually from a press box overlooking the track. The information was transmitted via telephone hook-up to a Level 66 computer in Billerica, Mass. that digested the raw statistics and instructed the terminal to generate a printout of the results.

Scoring was somewhat complicated, especially in the 10-kilometer race. This event often attracted 300 or more runners, of whom the last one to cross the finish line could conceivably determine the winning team. Standings changed after each race, so a prompt, accurate way of providing up-to-the-minute information had to be found. A computer was the answer.

#### *Yacht racing*

For the first time in the history of sailing competitions, computers and telecommunications were linked in 1979 to follow "live"—almost in real time—the progress of the different competitors in the first two-way transatlantic yacht race, TRANSAT. The race covers some 6,000 nautical miles (11,000 kilometers) from Lorient (France) to Bermuda and back again to Lorient.

The computer and telecommunications resources were provided by the ARGOS system of France's national space research center (Centre National d'Etudes Spatiales—CNES). ARGOS, a system for the collection of data by satellite, covers the entire globe. It is based on portable beacons (to collect and broadcast data), a TIROS-N satellite, and the data processing center of the CNES at Toulouse (France). In this center, a large-scale Cii Honeywell Bull IRIS 80 computer stored and processed information captured in two U.S. locations as well as in Lannion (Brittany), France. The data was re-transmitted to Toulouse by private land links.

Since the satellite's rotation around the earth allowed the positions of the competitors (to an accuracy of one nautical mile) to be transmitted to the IRIS 80 every two hours, it was possible to interrogate the computer at any instant for up-to-date information. Throughout the race, Cii Honeywell Bull, Honeywell's French computer associate, provided radio and TV representatives with special information services, including an automatic display of the paths of the yachts on a color graphic terminal. The display was generated by a Cii Honeywell Bull 66/60 computer in Paris from information supplied by the IRIS 80. It was then re-transmitted to the TV channel Antenne 2, which made regular use of it in its broadcasts on TRANSAT.

The major function of the 66/60 was to enable the TRANSAT control room to keep the state of the competition and the relative positions of the participants under continuous surveillance by furnishing information on the competitors' positions, distances covered, and standings in the various classes.

#### *Gymnastics*

Nadia Comaneci never had it so good.

When 81 of this country's finest female gymnasts recently competed against each other at the U.S. Gymnastic Federation's 1979 Junior Women's Championships in Allentown, Pa., a Honeywell Level 6 minicomputer was on hand to help both the participants and their coaches keep track of the scoring. Since the meet's winners were considered front-runners in the race for inclusion on the 1984 U.S. Olympic team, all the competitors—aged 10 to 14—were extremely anxious to learn where they stood in the overall scoring before and after each event.

There were two rounds of events—compulsory and optional—with four events in each round: balance beam, vault, uneven parallel bars, and floor exercises. Each event was evaluated by a panel of four judges, who had been former high-ranking amateur and/or professional gymnasts. The average of the judges' scores was first displayed to the audience and then entered into the Honeywell Level 6 Model 47 minicomputer by two terminal operators.

The system tabulated the scores and provided printouts of the results for the coaches. Reports detailing the standings of each event and the overall standings of each round were generated continuously during the three-day competition, so each competitor and her coach knew where she stood at all times. The top 10 competitors in each individual event qualified for the finals.

Nor is the computer's involvement with gymnastics limited to the U.S. At the 19th World Gymnastics Championships at Strasbourg, France in October of 1978, the data processing service that handled the thousands of scores awarded to some 400 athletes from 40 different countries was provided by Cii Honeywell Bull, Honeywell's French computer associate.

The role of the computer results service was to provide the 400 newspaper, TV and other journalists, as well as television viewers worldwide, with the unofficial intermediate placings as soon as each competitor finished his or her performance.

The data processing system, built around a Level 6 minicomputer, provided television producers with a continually up-to-date results table that was superimposed, as desired, on the competition images being broadcast. The calculated results were supplied in video signal form from a character generator connected to the computer.

The Level 6, operating in a real-time, multiprogramming mode, handled several competitions occurring at the same time in different locations. The placings were printed out every half-day on a 300 line-per-minute printer, reproduced and distributed; in total, some 150,000 documents were produced.

### *Bowling*

Small computer systems are freeing hundreds of bowling league secretaries of the drudgery of record keeping, hand-capping, recapping, calculating, posting and other chores.

The Brunswick Division, a leading supplier of bowling center equipment, configured its Integrated Retail Bowling Information System for bowling centers around Honeywell Level 6 computers. IRBIS is a functionally modular system that provides automation for tasks usually associated with the operation of a bowling center. The first module to be installed was for league record service. It administers awards programs, identifies absentees, maintains files of bowlers' names, addresses, and telephone numbers, and updates financial data. Each week, the system prints out a standings sheet that provides pertinent information on performances, individual player team standings, and lane assignments for each team.

### *Figure skating*

Two Cii Honeywell Bull computers played a key role at the 1979 World Figure Skating Championships, held in Vienna from March 13 to March 18.

Involving 350 participants from 26 countries, the championships were covered by some 500 journalists and television commentators. They were also broadcast by Eurovision, Intervision (covering Eastern Europe), ABC-TV of the United States, C.T.V. of Canada, and the television networks of Japan, Korea, the Philippines, and Mexico. A major feature of the official data processing results service was an information system for TV commentators provided by Honeywell Bull A.E., the Austrian member of Ciii Honeywell Bull's international network.

Two Cii Honeywell Bull 61/60 computer systems at the company's Vienna data processing center collected, via input terminals in the Vienna Stadthalle, the marks awarded competitors by the judges. The systems operated in parallel and, when necessary, each was switched to the other in seconds without any interruption to the results service. Provisional and final results, together with other information such as complete details of each competitor's performance, were printed on output terminals.

Other terminals were part of the TV commentator information system. A video screen displayed the first ten placings in the current competition, and optionally the second ten. These lists were instantaneously updated at the end of each competitor's performance. A closed-circuit TV installation conveyed the information to the control consoles of the TV units where it was selected for broadcast over Eurovision and other networks.

The programs for this computerized results service were essentially the same as those written by Cii Honeywell Bull for the 1976 Innsbruck Winter Olympic Games where the results service was also provided by Honeywell Bull A.G. They were modified to take into account competition rule changes, and a new program was added to provide additional information that facilitated the work of judges.

### *Air races*

For the dozens of pilots who flew their planes in the air race that started at Burke Lakefront Airport in Cleveland over Labor Day weekend last year, the real excitement started just a few seconds after the competition ended. By that time, thanks to Honeywell's DATANETWORK and a software program specifically designed to score air races, a Level 66 large-scale computer system located more than 500 air miles away in Minneapolis generated a cockpit-full of statistics, including a complete list of all entrants' times, speeds and order of finish.

Contestants in the Lake Erie Air Derby chose their own speed handicap in miles-per-hour and their own fuel handicap in gallons-per-hour. The purpose of the event was to measure the pilots' ability to fly a cross-country course according to their chosen speed and fuel handicaps. Final scores were determined by matching each contestant's actual speed and actual fuel consumption against the fuel handicap, with each category accounting for half the total.

If a pilot estimated that he would fly at 118 mph, for instance, but actually averaged 118.3, his proficiency rating would have been 99.97 percent. If he also predicted that his plane would consume 25.1 gallons during the race, but it actually used 24.8, his proficiency rating would have been 98.8 percent. By combining the two ratings and dividing the total by two, the system would award an overall score of 99.39 percent. Thus an individual who accurately predicted his exact speed but who consumed a lot less fuel than he anticipated would stand little chance of winning.

The winner in 1976, for example, was 12th in predicting her miles-per-hour, but second in estimating her fuel consumption. Her combined proficiency rating was the highest of any participant.

As soon as the last plane was "topped off" (refueled) at the end of last year's race, terminals sent the actual starting time, finish time and fuel consumption for all aircraft to the Level 66 computer in Minneapolis. The system already had on file the names of the pilot and co-pilot, the plane manufacturer or type, the plane number, the contestant number and the speed and fuel handicaps. Within minutes, the system digested the numbers and provided both proficiency ratings and final scores.

### *Basketball*

At 1979's National Basketball Association All-Star Game, hosted by the Detroit Pistons in the Silverdome in Pontiac, Mich., a Honeywell Level 6 Model 33 computer system enabled sportswriters to wait until the final minute of play before casting their ballots for the game's Most Valuable Player. The system sorted, tabulated, and simultaneously projected the results on two 24-inch CRT display stations, both of which were in full view of hand-held CBS cameras. Thus, the players, spectators and millions of television viewers knew the MVP within one minute after the game had ended.

The technique was even simpler than the method that worked so flawlessly at 1978's game in Atlanta, where a com-

puter was used for the first time to monitor the voting. During that game, a total of 20 Honeywell terminals were placed, in clusters of five, at various areas strategically located near the voting writers and sportscasters. With one minute left in the game and also at the final buzzer, volunteers collected the ballots and rushed them to the nearest terminal cluster where the votes were entered. Each cluster was connected via dial-up phones to a Level 66 large-scale computer in downtown Atlanta. The results were then projected on a CRT for the CBS camera.

The procedure in 1979 was considerably more glamorous in two respects. First, the collectors were members of the Detroit Pistons' Classy Chassis. Secondly, the voters did not have to enter a player's name on their ballot. Instead, they simply selected from either of two distinctly-colored packets in front of them—one for the East team and one for the West team—the card bearing the preprinted name of the player of their choice. A tabulating program operating within the Level 6 computer system read and totalled the results while "entertaining" the viewing audience with various screen images.

### *Baseball*

In this age of free agents, arbitration, non-negotiable contract demands, suits and counter-suits, one fact of baseball life remains constant: winning teams are those with the best 25-man rosters. To fill those rosters, major league owners and general managers use a variety of techniques, including trading for other players, outbidding the competition for the services of specific free agents, and, of course, scouting the minors for future Tom Seavers and Dave Parkers.

But scouting isn't what it used to be. Today, information on a prospect's hitting, throwing, running, and fielding abilities is just as likely to emanate from a computer as from the pen of a retired player working as a scout for his old club. In fact, 17 of the 26 major league teams—besieged by skyrocketing players' salaries, extremely expensive stadium maintenance costs, and assorted other rising expenses—have abandoned the old, costly system whereby each club maintained its own nationwide scouting organization. Instead, these clubs subscribe to the computerized scouting services offered by the Major League Scouting Bureau in Newport Beach, Calif.

The MLSB employs 60 professional scouts who watch baseball games across the country in behalf of their clients in both the National and American Leagues. The information they gather is stored in Honeywell's Computer Service, DATANETWORK. The large data base containing the player performance information is updated daily from the Major League Scouting Bureau's offices in Newport Beach. The client teams, using either video display or teletype terminals, simply dial into Honeywell's large-scale computer systems in Minneapolis and request the profiles of players by name, position, location or ranking.

During the season, scouts in five regions cover games played by high school, college and minor league teams. They rate promising players by height, weight, ability to play their position and batting skill. The qualifiable parameters are

combined with subjective comments on the players and subsequently sent to the MLSB main office where they are loaded offline onto a cassette, and then dumped online daily into the computers. DATANETWORK's dial-in access and simplified procedures allow non-technical users to process all this information.

Twice a year, reports are produced for each team on the 500 most promising high school and college players and mailed to the scouting director before the free agent draft in January and June. A similar report—much larger and more comprehensive—on all players in the minor league pro system is produced just after the draft in June and sent to each subscribing team, then updated throughout the minor league season.

Client teams and Scouting Bureau personnel, using interactive or batch terminals, have easy access to DATANETWORK in over 250 time-sharing cities located in the United States and Canada. When a team dials in and asks for updates, it receives all scouting opinions including second or third visit impressions by the same scout.

A printout on a particular player will include the latest preferential order relative to other players contained within the data base. The preferential order is based on all information available from scouts, who use a 2-to-8 rating scale for each category of performance.

Preferential lists for the free agent drafts contain all the players by position, scout, and state where they played. To make this list, players must have a total of 20 points minimum, with 80 points the maximum.

The reports for the pro system are large and more complicated. They assess 23 different aspects of a player's performance in such areas as running, throwing, hitting, fielding, accuracy, range and aggressiveness. An overall evaluation from all scouting reports is produced by the computers and added to the composite comments.

The reports are printed and arranged according to the farm system in which the players play. Only those players in a particular farm system are profiled for the parent major league club. Password and other identifiers are used to limit access to specific teams for information on individual pro system players.

To the Major League Scouting Bureau, DATANETWORK is indeed a Most Valuable Player.

### *Cricket*

Experience in the commentary box convinced ex-England cricket captain and BBC commentator Ted Dexter that the introduction of the computer as a statistical tool could be of importance both to the game and to the huge audience that follows it on television. His ideas were welcomed by one of cricket's most respected statisticians, Irving Rosenwasser who, together with BBC-TV producers David Kenning and Nick Hunter, began working with Honeywell Information Systems Ltd. to enlist modern computer technology for the benefit of cricket and its followers.

The outcome was a Level 6 Model 33 computer system, with 64K words of memory, cartridge disk and two visual display units, together with a special 'black box' designed

by Honeywell's Hemel development group to interface the Level 6 to the TV camera, thereby enabling the contents of the VDU screens to be duplicated on viewers' screens at home.

Under the control of Honeywell's MOD 200 operating system, the Level 6 stores score-cards and a wide range of cricketing statistical data. One of the VDUs is located in the commentary box, allowing an operator to request the display of relevant information, such as score displays, analyses, statistics, etc. This data is then simultaneously displayed in the commentary box and, if required, transmitted in the same format.

The screen format has been refined to provide two separate viewing sectors, one of which displays data duplicated on viewers' screens while the other—not available for transmission—contains additional information for the commentary box.

On-air trials were conducted during TV coverage of the Prudential World Cup, and the BBC and Honeywell have agreed to go ahead with computerized statistics for televised coverage of cricket.

## CONCLUSION

This concludes my list of the existing examples of sports-related computer applications, but I think you'll agree that their diversity and quantity are quite impressive. A similar discussion of the same topic a few years hence could—at least theoretically—touch on virtually any aspect of every sport currently pursued by man. If the trend continues at its current pace, we may be nearer than we think to the day that shouts of "Kill the umpire" will be replaced with "Debug that Model 2310."

### 83RD ANNUAL BOSTON MARATHON

APRIL 16, 1979

#### DISTRIBUTION OF RUNNERS BY AGE GROUPS

AGE	NUMBER ENTERED	NUMBER STARTED	NUMBER FINISHED	AVERAGE TIME	PERCENT FINISHED
UNDER 20	125	125	96	02:54:08	76.8
20-24	893	890	715	02:51:07	80.3
25-29	1489	1484	1162	02:52:10	78.3
30-34	1476	1470	1171	02:55:25	79.6
35-39	934	932	728	02:58:41	78.1
40-44	1671	1671	1216	03:12:46	72.7
45-49	869	869	600	03:15:15	69.0
50-54	338	337	209	03:19:01	62.0
55-59	93	93	49	03:17:38	52.6
60 & OVER	39	39	12	03:22:30	30.7
TOTALS	7927	7910	5958	03:01:15	75.3

THE AVERAGE RUNNER WAS 34.5 YEARS OLD

#### MALE: AVERAGE TIME BY WEIGHT GROUPS

WEIGHT	NO. OF RUNNERS	AVERAGE TIME
UNDER 100	3	02:56:48
100-109	3	02:55:02
110-119	42	02:50:08
120-129	259	02:52:28
130-139	930	02:55:20
140-149	1532	02:57:59
150-159	1452	03:02:43
OVER 159	1441	03:05:54
TOTAL	5662	03:00:28

THE AVERAGE RUNNER WEIGHED 149.3 POUNDS



## MALE: DISTRIBUTION OF FINISHERS BY HEIGHT AND WEIGHT

WEIGHT	UNDER 5'0"	5'0"-5'2"	5'3"-5'5"	5'6"-5'8"	5'9"-6'0"	OVER 6'0"	TOTALS
UNDER 100	0	1	1	1	0	0	3
100-109	1	0	2	0	0	0	3
110-119	2	0	13	27	0	0	42
120-129	17	2	57	148	33	2	259
130-139	60	3	47	466	348	6	930
140-149	73	15	11	404	974	55	1532
150-159	52	25	7	147	1073	148	1452
OVER 159	14	61	15	33	804	514	1441
TOTALS	219	107	153	1226	3232	725	5662

## FEMALE: AVERAGE TIME BY HEIGHT GROUPS

## MALE: DISTRIBUTION OF RUNNERS BY AGE GROUPS

AGE	NO. ENTERED	NO. STARTED	NO. FINISHED	AVERAGE TIME	PERCENT FINISHED
UNDER 20	106	106	83	02:51:05	78.3
20-24	807	804	658	02:49:09	81.8
25-29	1341	1336	1081	02:50:32	80.9
30-34	1346	1340	1097	02:53:58	81.8
35-39	851	849	685	02:57:31	80.6
40-44	1636	1636	1197	03:12:33	73.1
45-49	853	853	592	03:15:09	69.4
50-54	335	334	208	03:18:58	62.2
55-59	93	93	49	03:17:38	52.6
60 & OVER	38	38	12	03:22:30	31.5
TOTALS	7406	7389	5662	03:00:28	76.6

HEIGHT	NUMBER OF RUNNERS	AVERAGE TIME
UNDER 5'0"	54	03:18:44
5'0"-5'2"	37	03:14:40
5'3"-5'5"	105	03:17:13
5'6"-5'8"	84	03:15:30
5'9"-6'0"	14	03:11:23
OVER 6'0"	2	03:21:16
TOTALS	296	03:16:26

THE AVERAGE RUNNER WAS 34.8 YEARS OLD

THE AVERAGE RUNNER'S HEIGHT WAS 5'3"

## FEMALE: AVERAGE TIME BY WEIGHT GROUPS

WEIGHT	NO. OF RUNNERS	AVERAGE TIME
UNDER 100	14	03:16:33
100-109	70	03:14:27
110-119	113	03:17:36
120-129	61	03:16:40
130-139	28	03:15:59
140-149	10	03:17:09
TOTALS	296	03:16:26

THE AVERAGE RUNNER WEIGHED 115.2 POUNDS

## FEMALE: DISTRIBUTION OF FINISHERS BY HEIGHT AND WEIGHT

WEIGHT	UNDER 5'0"	5'0"-5'2"	5'3"-5'5"	5'6"-5'8"	5'9"-6'0"	OVER 6'0"	TOTALS
UNDER 100	4	8	2	0	0	0	14
100-109	9	19	33	9	0	0	70
110-119	19	10	55	28	0	1	113
120-129	11	0	12	36	1	1	61
130-139	9	0	3	11	5	0	28
140-149	2	0	0	0	8	0	10
150-159	0	0	0	0	0	0	0
OVER 159	0	0	0	0	0	0	0
TOTALS	54	37	105	84	14	2	296

## FEMALE: DISTRIBUTION OF RUNNERS BY AGE GROUPS

AGE	NUMBER ENTERED	NUMBER STARTED	NUMBER FINISHED	AVERAGE TIME	PERCENT FINISHED
UNDER 20	19	19	13	03:18:35	68.4
20-24	86	86	57	03:13:40	66.2
25-29	148	148	81	03:13:55	54.7
30-34	130	130	74	03:16:57	56.9
35-39	83	83	43	03:17:17	51.8
40-44	35	35	19	03:26:40	54.2
45-49	16	16	8	03:23:01	50.0
50-54	3	3	1	03:29:21	33.3
60 & OVER	1	1	0	00:00:00	0.0
TOTALS	521	521	296	03:16:26	56.8

THE AVERAGE RUNNER WAS 29.8 YEARS OLD

## AVERAGE TIME BY HEIGHT GROUPS

HEIGHT	NUMBER OF RUNNERS	AVERAGE TIME
UNDER 5'0"	273	03:04:13
5'0"-5'2"	144	03:04:31
5'3"-5'5"	258	03:07:24
5'6"-5'8"	1310	02:59:58
5'9"-6'0"	3246	03:01:01
OVER 6'0"	727	03:00:44
TOTALS	5958	03:01:15

THE AVERAGE RUNNER'S HEIGHT WAS 5'9"

## AVERAGE TIME BY WEIGHT GROUPS

WEIGHT	NUMBER OF RUNNERS	AVERAGE TIME
UNDER 100	17	03:13:04
100-109	73	03:13:39
110-119	155	03:10:09
120-129	320	02:57:05
130-139	958	02:55:56
140-149	1542	02:58:06
150-159	1452	03:02:43
OVER 159	1441	03:05:54
TOTALS	5958	03:01:15

THE AVERAGE RUNNER WEIGHED 147.6 POUNDS

## DISTRIBUTION OF FINISHERS BY HEIGHT AND WEIGHT

WEIGHT	UNDER 5'0"	5'0"-5'2"	5'3"-5'5"	5'6"-5'8"	5'9"-6'0"	OVER 6'0"	TOTALS
UNDER 100	4	9	3	1	0	0	17
100-109	10	19	35	9	0	0	73
110-119	21	10	68	55	0	1	155
120-129	28	2	69	184	34	3	320
130-139	69	3	50	477	353	6	958
140-149	75	15	11	404	982	55	1542
150-159	52	25	7	147	1073	148	1452
OVER 159	14	61	15	33	804	514	1441
TOTALS	273	144	258	1310	3246	727	5958

## MALE: AVERAGE TIME BY HEIGHT GROUPS

HEIGHT	NUMBER OF RUNNERS	AVERAGE TIME
UNDER 5'0"	219	03:00:39
5'0"-5'2"	107	03:01:01
5'3"-5'5"	153	03:00:39
5'6"-5'8"	1226	02:58:54
5'9"-6'0"	3232	03:00:58
OVER 6'0"	725	03:00:40
TOTALS	5662	03:00:28

THE AVERAGE RUNNER'S HEIGHT WAS 5'9"

STATE/COUNTRY	NO. ENTERED	NO. ENTERED	PCT. STARTED	NO. FINISHED	PCT. FINISHED
Australia	6	5	83.3	3	60.0
Belgium	2	2	100.0	1	50.0
Bermuda	17	17	100.0	16	94.1
Brazil	1	1	100.0	0	0.0
Canada	310	310	100.0	235	75.8
Columbia	4	3	75.0	3	100.0
Costa Rica	4	4	100.0	3	75.0
Denmark	2	2	100.0	1	50.0
England	36	36	100.0	24	66.6
Egypt	1	1	100.0	0	0.0
Finland	6	6	100.0	5	83.3
France	3	3	100.0	0	0.0
Germany	13	13	100.0	6	46.1
Greece	1	1	100.0	1	100.0
Guam	1	1	100.0	1	100.0
Haiti	1	1	100.0	1	100.0
Holland	1	1	100.0	1	100.0
Ireland	19	18	94.7	9	50.0
Israel	1	1	100.0	0	0.0
Japan	46	46	100.0	31	67.3
Kenya	2	1	100.0	1	50.0
Korea	1	1	100.0	1	100.0
New Mexico	1	1	100.0	1	100.0
New Zealand	12	12	100.0	9	75.5
Norway	1	1	100.0	1	100.0
Philippines	2	2	100.0	2	100.0
Portugal	1	1	100.0	0	0.0
Puerto Rico	18	18	100.0	12	66.6
Saudi Arabia	2	2	100.0	0	0.0
Scotland	1	1	100.0	1	100.0
Sweden	36	36	100.0	23	63.8
Switzerland	1	1	100.0	0	0.0
Turkey	1	0	0.0	0	0.0
U.S.A.	1	1	100.0	1	100.0
United Kingdom	1	1	100.0	1	100.0
Wales	1	0	0.0	0	0.0
West Germany	1	1	100.0	0	0.0
none	1	1	100.0	1	100.0

STATE/COUNTRY	NO. ENTERED	NO. ENTERED	PCT . STARTED	NO. FINISHED	PCT . FINISHED
Alaska	14	14	100.0	9	64.2
Alabama	40	40	100.0	31	77.5
Arkansas	11	11	100.0	9	81.8
Arizona	96	96	100.0	66	68.7
California	848	847	99.8	566	66.8
Colorado	112	112	100.0	85	75.8
Connecticut	278	278	100.0	222	79.8
Dist. of Columbia	64	64	100.0	46	71.8
Delaware	31	31	100.0	23	74.1
Florida	192	190	98.9	140	73.6
Georgia	122	122	100.0	95	77.8
Hawaii	28	28	100.0	17	60.7
Iowa	50	50	100.0	37	74.0
Idaho	9	9	100.0	6	66.6
Illinois	213	213	100.0	142	66.6
Indiana	110	110	100.0	76	69.0
Kansas	19	19	100.0	14	73.6
Kentucky	45	45	100.0	38	84.4
Louisiana	49	49	100.0	36	73.4
Massachusetts	994	990	99.5	764	77.1
Maryland	215	215	100.0	172	80.0
Maine	70	70	100.0	57	81.4
Michigan	236	236	100.0	178	75.4
Minnesota	134	134	100.0	94	70.1
Missouri	59	59	100.0	42	71.1
Mississippi	18	18	100.0	15	83.3
Montana	8	8	100.0	6	75.0
North Carolina	119	119	100.0	99	83.1
North Dakota	11	11	100.0	10	90.9
New Hampshire	90	90	100.0	68	75.5
New Jersey	319	318	99.6	251	78.9
New Mexico	42	42	100.0	30	71.4
New York	904	903	99.8	695	76.9
Nebraska	14	14	100.0	13	92.8
Nevada	9	9	100.0	9	100.0
Ohio	249	249	100.0	205	82.3
Oklahoma	10	10	100.0	9	90.0
Oregon	78	78	100.0	53	67.9

STATE/COUNTRY	NO. ENTERED	NO. ENTERED	PCT . STARTED	NO. FINISHED	PCT . FINISHED
Pennsylvania	426	424	99.5	332	78.3
Rhode Island	105	105	100.0	84	80.0
South Carolina	49	49	100.0	38	77.5
South Dakota	12	12	100.0	8	66.6
Tennessee	75	75	100.0	60	80.0
Texas	203	203	100.0	155	76.3
Utah	27	27	100.0	24	88.8
Virginia	237	237	100.0	179	75.5
Vermont	58	58	100.0	46	79.3
West Virginia	37	37	100.0	31	83.7
Washington	84	84	100.0	57	67.8
Wisconsin	140	140	100.0	117	83.5
Wyoming	5	5	100.0	4	80.0



# Computers helping dance notation help the dance: a vision

by STEPHEN W. SMOLIAR

General Research Corporation  
Santa Barbara, California

## 1. DANCE NOTATION: WHAT AND WHY?

The production of a ballet is one of the most frustrating endeavors in the performing arts today. The frustration stems from the fact that a substantial amount of information must be shared among a large number of individuals, and the only manifestation of this information is in a few human memories. Often the information is evolving: a choreographer will work from day to day with a company of dancers, saving only a fraction of material from one day to the next, until, eventually, the "vision" of a complete piece of choreography has been formed. Alternatively, in the case of reconstructing a piece of choreography, disagreements inevitably arise as to whose memory of the original is most accurate. In the absence of any "hard" information, such disagreements can only be resolved by the strength of authority.

These problems do not arise when a symphony orchestra prepares a concert. In fact, such problems are quite unthinkable in the world of music. This is because the "vision" of the composer has been set down in a notation which has been second nature to the vast majority of performing musicians for well over a thousand years. However many years he may have been lying in his grave, the composer has managed to communicate his authority to the performers of today through the score and part books of his music.

The predominance of music notation has led many "fans" to assume that notation plays a similar role in the dance. Unfortunately, this is only a half-truth. In fact, the origins of dance notation go back practically to the origins of classical ballet ([Hutchinson]); but, as we shall see, notation has never "caught on" among dancers as it did among musicians. In the following section we shall attempt to analyze why this is the case, after which we shall consider how the computer might be able to remedy this situation.

## 2. CURRENT PROBLEMS IN DANCE NOTATION

Any idea which is unpopular always has a bastion of myths to support its unpopularity. The primary myth about dance notation is that *it can't possibly work* (not that it *doesn't* work, mind you—one may simply deny the possibility of contradicting evidence). The reason behind this myth inevitably stems from an argument to the effect that the human body has so much more subtlety and so many more degrees

of freedom than any musical instrument that no notation could ever come close to capturing such an overwhelming amount of information. Leaving the dance community aside for a moment, such an argument would be regarded as patent nonsense by any performing musician. He knows that his notation does not embody the full range of subtlety of expression on an instrument; indeed, that is what makes performing so interesting. He understands that the score is but an *abstraction* of a musical performance and that performance is unthinkable unless he first contributes a substantial amount of his own information to that score. Are we to assume, then, that no such level of abstraction exists for choreography? *Au contraire!* History has provided us with an abundance of abstractions, and *this* turns out to be one of the more substantive problems surrounding dance notation.

### 2.1 Lack of universality

In the early years of ballet, dance notation was not a particularly burning issue because it was a rather simple matter. All dances were made out of a relatively small number of archetypal patterns, and "recording" a dance was simply a matter of indicating which patterns were selected, in what order they were executed, and what path the dancer followed while executing these patterns. (An analogy with the neumatic notation of chant may be appropriate.)

As the vocabulary of ballet became freer, such "neumatic" notations became less useful. The issue of "commonly accepted patterns" also dissolved as dance styles began to cross international boundaries. There followed a wide variety of attempts to record movement iconographically. (The number of variations on the stick figure in the name of dance notation is almost mind-boggling.) Unfortunately, such "icons" could never represent *movement*; they could only represent selected positions assumed in the course of movement. *How* one progressed from position to position tended to be described in an *ad hoc* manner, generally fully understood only by the inventor of the notation.

In spite of these many unsuccessful attempts, this century has seen two genuine abstractions of human movement incorporated into notations—one developed by Noa [Eshkol] and Abraham Wachmann, the other by Rudolph Laban ([Hutchinson]). Both of these abstractions are based on the skeletal system—a view of the body as a system of bones



connected at joints. Both also incorporate systematic representation of the passage of time. Thus, one is presented with a continuous representation of positions assumed by the skeleton throughout the flow of time, as opposed to the "selected snapshots" of an iconographic notation.

Unfortunately, while these two notations share a common abstraction, their syntaxes differ radically. Neither can be readily embraced by one who is familiar with the other. This, then, is the key "political" problem with dance notation. Each notation has its own strongly devoted band of followers, organized as an international society and firmly convinced that theirs is "the true way." At a time when it is hard enough to get the majority of the dance community to accept notation of *any* sort, such factionalism is of little benefit. (Incidentally, several of the iconographic notations have also managed to gather their own factions. A dancer who is seriously interested in notation is bound to have about as much trouble as a Republican who is seriously interested in a presidential candidate.)

In a sense, one may say that the presence of two viable notations is worse than having none at all. Excessive quibbling over syntax tends to cause one to forget that at the foundation of both is an excellent semantic model for describing choreography. While it would not be particularly difficult to train a dancer to read *both* notations, there being no differences in the basic principles, the antagonism of factionalism will continue to discourage any dancer from learning *either*.

## 2.2 Difficulties in recording

Once a notation is selected, one must still face the fact that preparing a dance notation score is not an easy process. The main difficulty is that while a composer may be able to get all his ideas set down in score working strictly on his own, a choreographer tends to grow his ideas out of interactions with his dancers. Under such circumstances, a choreographer is not really in a position to spend his time writing scores; so this role is assumed by a third party, a "dance notator" who acts somewhat like a court stenographer while rehearsals are in progress.

A professional notator described the difficulties in preparing a dance notation score as follows ([Brown]):

"First of all, before the notator's preparation of the final pencil draft, there is the process of writing and rewriting rough drafts. As the dancers learn, the notator jots down symbols. If there are many dancers quickly learning difficult movements, it becomes impossible for the notator to write everything while the dancers are learning. The dancer learns a total movement with all parts of the body operating "in parallel," while the notator must record each change in a body part—being limited by the speed at which he can write. Because of this limitation, the notator learns to write essential key symbols which cue his memory. When viewing his notes after the rehearsal is over, the notator will fill in the details and check these at the next day's rehearsals.

A problem with this way of working arises when a notator works with a large ballet company staging a new work. Because

of the company's organization, many hours a day must be devoted to teaching the work. In some situations, the choreographer may be working with different groups of dancers throughout a ten to twelve hour period. This makes the process of filling in details "after hours" difficult. As a rule, the filling-in process generally requires one to two hours for each hour spent in rehearsal. Even if we overlook the fatigue of a day which involves twelve hours of rehearsal time, simple arithmetic shows that there just aren't enough hours in the day to keep up with the work outside rehearsal. The notator must be tremendously organized and must have sufficient stamina to stay abreast of what every dancer in the company is learning—committing most of this information to memory. Frustration sets in when the choreographer decides to add and drop parts of the dance or to revise steps and sections. Battling constantly with the organization of notes, the notator tries to "get everything down," filling in missing spots during "clean-up" rehearsals, which are conducted after all the dancers have learned their parts. Finally, when the dance is ready for production, the notator collects all the information regarding props, scenery, lighting and costumes. These are included in the score, since the score serves as a historical document to be used for reconstruction purposes.

Under such working conditions the notator is in no position to work on the final pencil draft as the rough notes are accumulated. The notator will not have such time until after the dance is in performance. The job then becomes a matter of many hours of solitary work copying the rough notes, laying out pages of graph paper, and refining the actual notation used.

Autography, in itself, is also a very time-consuming process. The autographer must know enough Labanotation to be familiar with the symbols to be copied and the basic rules of layout (in case the layout of the final pencil draft has to be modified). When the autography is done by hand, the autographer must also be skilled in working with special pens, indelible ink, templates, and reproducing paper. The symbols are arranged on the page as specified in the notator's pencil draft and layout booklet. Calculations for margins and aesthetic spacing are made before proceeding with the inking, where each line is separately drawn and connected."

It should be noted that the problems of autography are more serious for Labanotation than they are for the notation of Eshkol and Wachmann. However, the basic problem of collecting the actual data remains the same. Furthermore, the astute reader will have noted in the above scenario that one must still rely heavily on the powers of human memory. Once a score is prepared, it can bear the weight of authority; but the problem of establishing that a score is an *accurate* recording of the choreography is not a minor one. In general, the choreographer will not understand the notation well enough to pass judgment on it. This brings us to our third major problem.

## 2.3 Difficulties in reading

Unfortunately, the only individuals who are capable of *reading* dance notation scores are those same individuals who serve as dance notators. Neither choreographers nor dancers, in general, can, on their own, extract all the information which these scores contain. Not only does this mean that the choreographer is in no position to pass judgment on

the accuracy of the score, but also it implies that for purposes of reconstruction, a dance notator must again be brought to rehearsals. However, the role of this notator has now shifted from "court stenographer" to "ballet master." Using the score, the notator can *demonstrate* all the movements as they have been recorded. Thus, the notator will work with the dancers the same way that the choreographer does. The difference is that all authority of information resides in the score; the notator is simply the medium by which the dancers may gain access to the score.

### 3. COMPUTER ASSISTANCE

#### 3.1 Data entry

The above description of the plight of a dance notator in preparing a score might, in another context, be construed as an advertisement for a word processing system. In fact, the technology of word processing is precisely the sort of remedy which will alleviate all the time-consuming frustrations of score preparation. The only potential obstacle arises from the fact that the basic data structure for word processing is one-dimensional, the character string, while dance notation scores are inherently two-dimensional.

Fortunately, the technology of computer graphics allows us to manipulate two-dimensional structures as easily as one-dimensional ones. The real issue is whether or not the notation is well enough structured that it can be conveniently manipulated in a syntax-directed fashion. The notation of Eshkol and Wachmann poses no problem in this respect, since it consists of elementary configurations of numbers placed within the boxes of standard graph paper. Labanotation, on the other hand, has a much broader vocabulary of graphic symbols; but it has been demonstrated that these symbols are highly organized according to a structure which may be reflected in the internal structures of a text processing system ([Smoliar]).

Given such a "notation processing system," one may envisage the notator of the future working with a portable graphics terminal which may be easily installed in a ballet studio. All initial notes may be entered during rehearsal into some common file structure which will become the data repository for the particular ballet. As the steps are taught in greater detail, the notator will be capable of using the time to update the score as it has been recorded thus far. Then, as rehearsal enters the final stages, the notator will be able to follow along in the score, confirming the accuracy of the recorded material. Much of the routine efforts of "filling in" and "cleaning up" may be relegated to system functions performed by the notation processor, leaving the notator free to worry primarily about the semantic content of the notation. Finally, given a suitable device for hard-copy output, the need for a separate autography stage will be eliminated. All information necessary for graphic formatting will already be present in the file structure of the score, so that the preparation of a "final" draft will simply be a system output function.

How realistic is this vision? For Labanotation the basic theoretical problems have been solved, and a prototype system has been implemented ([Smoliar]). However, the technology of the implementation is not at all appropriate to the ballet studio, utilizing large and expensive processing and graphics equipment. The major problem is to take the results of research conducted to date and pass them through a development phase which would result in a usable product. Unfortunately, such a development project would entail a substantial expense for a product which would never be particularly widely used. (Even assuming an overwhelming interest in dance notation, the number of notators will never approach the number of secretaries.) Thus, the development of the product itself could never be cost-effective; and, as a result, for sheerly economic reasons, the feasibility of this vision is pathetically low.

#### 3.2 Notation interpretation

From a point of view of data processing, the problem of "notation illiteracy" is far more substantial than the data entry problem. The latter is only concerned with formatting a well-defined system of symbols; the former must address the semantics behind these symbols. Of course, the real issue behind these semantics is the issue of human movement itself. What is required, at the data processing level, is the ability to construct a simulator of human movement. Given the existence of such a simulator, one may then regard a notation score as a set of commands to that simulator.

How might such a simulator be structured? Clearly, the basic underlying model of the human skeleton must be present, since this model is incorporated in the abstractions of the systems of both Laban and Eshkol and Wachmann. When one addresses more detailed specifics, however, one discovers that much of the basic structure of Labanotation may be interpreted as a rather powerful plan for a highly general simulator ([Weber]). Under this model every joint of the skeleton may be regarded as being endowed with the "processing power" to orient itself with respect to some well-defined system of reference. Furthermore, both the origin and the axis-orientation of this system of reference may vary during the course of the simulation, according to specific commands incorporated into the notation. Furthermore, every moment is classified as either a *gesture*, which simply changes the orientation of the skeleton, or a *support*, which entails a major movement of the center of gravity. (Gestures will, necessarily, entail *minor* movements of the center of gravity.) A simulator based upon these principles has, in fact, been designed ([Badler]).

Clearly, no dancer or choreographer will be interested in the specific mechanisms of a computer simulation of human movement. However, given an implementation of such a simulator, one could monitor its behavior through a graphic display of human figures. These figures need not necessarily resemble "ideal" dancers. (Remember, the notation itself is still only an abstraction of the movement it represents.) However, the display should be capable of capturing all information which the notation has recorded. Furthermore,

given the essentially invariant behavior of the simulator, the display may be highly flexible. One may "observe" the simulator from alternative points of view, perhaps changing the point of view while the display is in progress. Given many figures, one may wish to ignore displays of all but one or two. These are facilities by which a dancer would be able to observe the steps and learn a part in a manner similar to the protocols of the rehearsal studio. (These facilities are also far beyond the capabilities of any conventional video recording techniques.) Finally, the output as prepared by such a simulation system would be a playback of the score which the choreographer could observe as a means of approving the accuracy of the contents of that score. Working in conjunction with the notator, the choreographer could have a direct hand in establishing the score's validity.

Once again, there is the question of feasibility of such a vision. If the prospects of a notation processing system are slim, there seems to be little hope for the development which would be necessary to produce such a display system which could become a convenient installation in a ballet studio. One can only hope that the need for the simulator itself may attract the interests of better-endowed institutions. It would not be the first time the arts would benefit from a "spin-off" of a product of high technology.

#### 4. OTHER VISIONS

##### 4.1 *An information management system for recorded dance*

###### 4.1.1 A network of data bases

Given that the two visions proposed in the preceding section appear to be rather remote, it may be somewhat unwise to fantasize further. Nevertheless, these visions have some implications which are worth dwelling upon in their capacity to foster further visions. The mere fact that we have proposed a variety of digital representations of human movement, for example, leads us to consider possible applications involving data bases.

One of the greatest problems facing the dance world, as we have seen, is the reliable dissemination of information. In general, a company learns its choreography from a choreographer who "resides" there. If another company wishes to perform the same ballet, they must make arrangements for the choreographer, or some other reliable authority, to "visit" for the purpose of teaching the choreography. While the choreographer may be the only reliable authority in the matter of teaching all the subtleties of performance, teaching the basic steps to a new company is generally a rather tedious and tiresome undertaking. The problem, once again, is one of the information imprisoned in an individual's memory.

A network of data bases of dance notation scores would go a long way toward alleviating this difficulty. The result would be one of a nationwide (if not worldwide) library of the ballet repertoire. Local sites would be responsible for recording and maintaining their share of this repertoire.

Given the ability for computer interpretation of dance no-

tation, this library would be accessible even to those "illiterate" in the notation. One could draw upon the computer not only to provide the score but also to provide the sort of performance of the score described in the previous section. Under these circumstances, a choreographer could rely upon the services of the computer to handle teaching the basic steps, leaving him free to concentrate upon the final details of performance. (A similar situation, without the use of a computer but with dancers "literate" in Labanotation, currently exists in the Syracuse Ballet [Ubell].)

###### 4.1.2 Copyright issues

A critical component of any information management system is a mechanism which protects the information managed by the system. Such a mechanism should secure responsibility for the creation of a score, as well as protecting the choreographer's rights to determine who may read that score. These ideas deserve a bit of further elaboration.

First of all, what is the nature of the information to be subject to protection? In the system proposed in the preceding section, this information is divided into two categories: (1) notation scores, and (2) animated interpretations of notation scores. Four levels of protection may be applied to both of these categories:

1. No access—a user is denied any access to a particular score or animation.
2. Read only—the user may use the system to view a particular score or animation, but access is limited to what may be observed while seated at a display terminal.
3. Copy—the user may request a physical copy of a score or a film or videotape of an animation.
4. Update—the user is allowed to modify the information in a notation score or animation or access the program which translates notation into animation.

The information management system may then maintain records regarding which levels of protection apply for which notation scores and animations to which users of the system. Authority to update will incorporate authority to change a particular protection level. Thus, initially, the notator will maintain a protection level of 4 and use it to limit access to preliminary versions of the score. (The choreographer and dancers, for example, may be allowed read only access, while the rest of the users are forbidden all access.) Ultimately, by assigning a protection level of 4 to a choreographer for a completed score and animation, the system automatically allows that choreographer to be the ultimate arbiter of protection status for his "personal" information or to delegate this authority to any user of his choice. Requests from users for permission to see protected information may also be handled by the system through a "mailbox" facility.

Under such a system it is likely that information will be better protected from copyright abuse than printed scores or films. A page of dance notation displayed at a terminal

cannot be taken over to a Xerox machine to be copied without authorization. Also, it should not be difficult to guard against users photographing or filming images displayed at a terminal. There remains the problem of a user copying out a notation score by hand, but this problem is comparable to that of an individual preparing a notation score strictly from attending performances of a ballet. While this system would not prevent all forms of copyright abuse, it would only allow those which are extremely difficult or inconvenient to implement.

#### 4.2 Choreography

While many choreographers derive their inspirations from spontaneous interactions with their dancers, not all choreographers enjoy the luxury of a company of dancers existing purely to satisfy their creative urges. Even in the best of companies, rehearsal time is limited; and a choreographer cannot always explore his creative urges at his personal convenience. Here, again, the composer is at an advantage. In the absence of an instrumental ensemble, he may still turn to a keyboard to experiment with his ideas.

The sort of facility we have been discussing could ultimately serve as an analog to the keyboard for a choreographer. Of course, it would require the choreographer to learn the notation; but is that asking more than requiring that a composer possess certain keyboard skills? The intent is not to use the computer to produce an artistic object, but rather to assist in those mental processes which are invoked during the act of creation. It will be little more than a device with which the choreographer may better plan his rehearsal time.

Considering what has happened in music, one must envisage the possibility of attempts to automate choreography itself. The results in music, to date, have been rather unimpressive. There have already been some analogous attempts in choreography. Unfortunately, in both cases these tend to be diversions of individuals who are rather casual practitioners of the art. Lacking the patience to negotiate the excruciating details of creation, they turn to technology for a

crutch. Unfortunately, *human* audiences tend to respond to acts of *human* creation; and unless the performers manage to contribute some element which transcends the meagre bookkeeping of the alleged choreographer, the resulting product tends to have little to offer even the most dedicated audience of human beings.

#### 4.3 Scholarship

If the production of dance appears to be in the Dark Ages when compared with the world of music, the issue of dance scholarship is practically pre-historic. Once again, the problem is one of recording information. It is very difficult to analyze a ballet when all one has are verbal accounts of that ballet. (One might just as well pass legal judgment strictly on the basis of hearsay evidence.) The accumulation of a repertoire in scores would open the doors to possibilities for comparative analysis. Even in music, the theory of composition, as we know it, did not come into its own until notation was a common practice. The study of dance history *can*, eventually, become more than an accumulation of indirect accounts, but only if we see to it that the dancers themselves are allowed the benefit of objective recording.

#### REFERENCES

- [Badler] Badler, N. I. and Smoliar, S. W., "Digital Representations of Human Movement," *Computing Surveys*, Vol. 11, pp. 19-38 (1979).
- [Brown] Brown, M. D., Smoliar, S. W., and Weber, L., "Preparing Dance Notation Scores with a Computer," *Computers and Graphics*, Vol. 3, pp. 1-7 (1978).
- [Eshkol] Eshkol, N. and Wachmann, A., *Movement Notation*, Weidenfeld and Nicolson, 1958.
- [Hutchinson] Hutchinson, A., *Labanotation*, Theatre Arts Books, 1970.
- [Smoliar] Smoliar, S. W. and Tracton, W., "A Lexical Analysis of Labanotation with an Associated Data Structure," *Proceedings 1978 Annual Conference: Association for Computing Machinery*, Vol. 2, pp. 727-730.
- [Ubell] Ubell, E., "Dance Notation Steps into a New Era," *The New York Times*, Sec. 2, pp. 12, 19, October 24, 1976.
- [Weber] Weber, L., Smoliar, S. W., and Badler, N. I., "An Architecture for the Simulation of Human Movement," *Proceedings 1978 Annual Conference: Association for Computing Machinery*, Vol. 2, pp. 737-745.



# Automatic Camera Effects System (ACES)

by STEVEN N. CRANE and R. DAVID SNYDER

*Walt Disney Productions*  
Burbank, California

## INTRODUCTION

The need to achieve realism in theatrical motion pictures with miniature models requires precise positioning, close tolerances, and a high degree of repeatability. A particular movement of the camera and the subject(s) may be repeated several times to create mattes or other special effects involving multiple exposures. Traditionally, miniatures photography has been performed "by hand." Each frame would be set up and shot individually, a tedious and time consuming procedure. Effects work of this sort was very expensive and required special skills and lots of patience; consequently, very little of this type of filming was done. The introduction of electronic, and recently computerized, motion control has tremendously advanced the state of the art. It is now possible to create effects relatively easily and inexpensively that a few years ago would not have been feasible. Electronic motion control enables the film maker to control the orientation of the camera and subject to a high degree of precision and move the system through several degrees of freedom.

Computerized motion control places the electronic motion control system under the control of a computer. The computer is programmed to provide the variety of specific types of control that the film maker will want to use. He may call up those controls, specify whatever parameters tailor the control mode to the particular instance, and the computer operates the motion control system to achieve the desired effect. The computer can also aid the film maker in other ways such as scene planning and storage and retrieval.

## EARLY WORK AT DISNEY

The first automated camera system at Disney was constructed in late 1970 by Ub Iwerks. It was designed specifically for filming passes over held art work for the "Hall of the Presidents" show at Walt Disney World. It consisted of a 30 foot long overhead track upon which the camera was mounted (the "truck axis"). A large square plate at one end of the track held the art work to be photographed. The plate could be moved vertically (north/south) and horizontally (east/west). Autofocusing was achieved with a cam driven off the truck axis. The system thus controlled a total of three axes.

The system was not computer operated. It would scan a strip of white paper. When a black line was detected on a particular "track" the axis controlled by that track would begin to move until another line was detected. When all axes halted, the shutter would be tripped automatically and the system would move on to the next frame. It operated in stop motion mode only. Unfortunately, the project got a late start, there were problems with sensitivity and consistency and with the deadline looming ever closer it was decided to resort to traditional methods to finish the job. Ub planned to convert from the white paper tape control method to punched paper tape when time became available. He also formulated plans to build an automated cartoon crane. Unfortunately, Ub died before he could implement any of his plans. With Ub gone there was no one left in a position of authority to push for innovative methods.

Several years later (1976) the studio faced the prospect of producing a science fiction movie (eventually dubbed "The Black Hole") involving hundreds of matte paintings and a tremendous workload of special effects processing. Without an equally tremendous increase in productivity the film could not be made. The idea of shooting held art (such as a matte painting) automatically was reexamined by Dave Snyder, Manager of Scientific Programming, Don Iwerks, machine shop manager, and matte artist, Harrison Ellenshaw. They obtained approval for constructing the "Matte-SCAN" system.

The Matte-SCAN system is a high precision numerically controlled stop-motion camera crane. The system moves along a 71 inch truck axis, with 50 inches east/west, and 26 inches north/south. Positioning accuracy is a thousandth of an inch in each axis. The camera is capable of exposure times from infinity to shutter speeds of over 24 frames per second. The system also provides automatic focusing. It is used to photograph a matte painting or a projected image. The projector frame advance is controlled automatically. There is also an auxiliary output channel which can be used for special effects devices to be synchronized with the operation of the rest of the system. The entire system is operated automatically by a programmable controller designed to convert non-n/c machine tools to numeric control.

Early tests revealed the Matte-SCAN system to be an unqualified success. This lent new credibility to the concept of computer-aided film making. Proposals first made by

Dave Snyder in late 1971 to build an automated motion picture camera were reexamined. Also, other motion control systems then in existence at other studios were considered. However, none of these provided exactly what was needed. It was realized that such a facility would become a permanent tool as basic as the Mitchell camera and Chapman crane to future film-making at Disney. Therefore, it was decided no reasonable expense would be spared in doing a thorough and complete job.

## DEVELOPMENT OF ACES

The go-ahead to build the Automatic Camera Effects System was received in early February 1978. The system was designed between January and May 1978. An "Operator's Manual" was issued in April 1978. This served as both a functional specification for the software design, and as a device for studio personnel (who would operate the system) to tailor the device to their needs. Software coding began on the operator interface that same month. As changes were made to the manual they were implemented in a demonstration program. The software was coded initially on a NOVA 2 system since the ACES computer was not ordered until May 1978, and not delivered until October. At this time production on "The Black Hole" was scheduled to begin August 1. Meeting that date would be impossible but every effort was made to keep the pace of development at a high level.

The software installed on the NOVA 2 included provisions for responding to commands as though the hardware were present. This enabled all functions but the interface to the hardware to be debugged before the hardware even existed. The servo equipment was ordered from an outside vendor in May 1978 and was delivered in October. The computer was interfaced to the servo controllers by Walt Disney World personnel. The first test shots were filmed on November 6, 1978, and the first production shots were filmed on November 24, 1978. After some light leakage problems in the camera were fixed, production began in earnest on December 13, 1978. Production of "The Black Hole" by the second unit was wrapped on October 12, 1979. In the interim the software had gone through several phases of enhancements and additions. The hardware was also continually being updated and improved.

## CONFIGURATION OF ACES

The Automatic Camera Effects System is permanently installed on Sound Stage 3 at Walt Disney Studios in Burbank, California. It consists of a camera stand mounted on a 68 foot long fixed two-rail track (Figures 1 and 2), a model stand riding a portable 30 foot track (Figures 3 and 4), and a minicomputer and servomotor controllers located in an air-conditioned computer room within the stage. Other devices are also used in conjunction with ACES including a process projector, a video camera, two video monitors and a video tape recorder, a 16 foot by 25 foot high frequency blue

screen, a 38 foot by 72 foot DC blue screen, and other effects devices that may be wired up to ACES to achieve some special purpose.

The computer is a Data General NOVA 3/12 with hardware multiply/divide, hardware floating point, 64K words of memory, a 2.5 megabyte cartridge disk, dual floppy disks, a Dasher video display terminal, and a Dasher printer terminal (see the block diagram in Figure 5). The ACES software operates under the Mapped Real Time Disk Operating System and is coded about 90 percent in FORTRAN V and 10 percent in NOVA assembly language.

The system operates in the foreground on the CRT terminal. The software consists of several concurrent tasks. The main task processes all user input and operates the hardware during filming. Other tasks display the current camera and model stand positions, monitor the digital inputs, and handle the jogging controls.

The camera stand has six degrees of freedom. The axes are identified as follows: truck (parallel to the track), east/west (horizontal, perpendicular to the track), north/south (vertical), pan (yaw), tilt (pitch), and roll plus focus. These axes are illustrated in Figure 1. The shutter and film transport are also computer controlled as to position and speed of advance/rewind. There is also an electronic shutter which is used for capping. The model stand has four degrees of freedom: truck (parallel to the portable model stand track), yaw, pitch, and roll. These axes are illustrated in Figure 3.

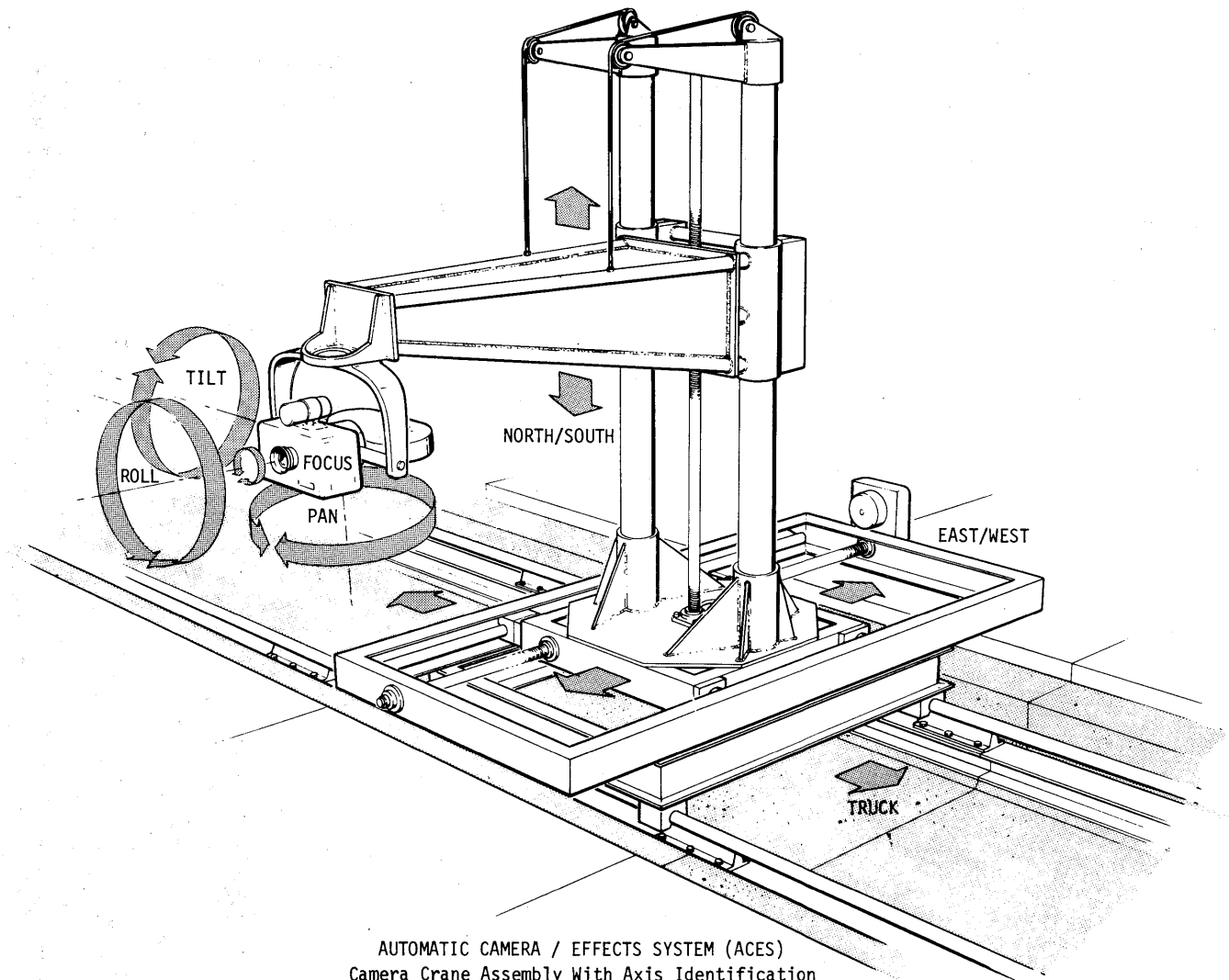
The resolvers which measure position are designed to provide a control tolerance of 0.01 inch for linear axes and 0.01 degree for rotational axes. The camera and model stand axes can be moved into a position which is within these tolerances of the desired position. Since the mechanical tolerance of the equipment is finer than the resolution of the resolvers, the hardware achieves even greater accuracy in repeating a movement through a sequence of positions.

## OPERATION OF ACES

ACES is designed to be operated by studio personnel, experts in film making but completely unfamiliar with operating a computer. A set of approximately 40 commands is entered via the keyboard. A row of special function keys above the keyboard is used to jog the camera stand or model stand into a desired position. The operator specifies a set of between two and twenty positions in order to create a take.

A take is the building block of a motion picture. It is a continuous strip of film which presents one unbroken sequence of action from one viewpoint. Each attempt to film such a sequence is a separate take, and several attempts are usually required to obtain a satisfactory result.

The positions specified by the operator are called key positions or station points. The camera stand and model stand are required to be at a particular key position at a specific frame number. A listing of key positions for one particular take is shown in Figure 6. The position of the camera and model axes for the frames between station points is obtained by a piece-wise cubic polynomial interpolation algorithm (the "inbetweeners"). The interpolation is done automati-



AUTOMATIC CAMERA / EFFECTS SYSTEM (ACES)  
Camera Crane Assembly With Axis Identification

Figure 1—ACES camera stand.

cally by the computer whenever a take is to be filmed or rehearsed. The inbetweeners create a file on the cartridge disk containing the desired camera positions at each frame in the take. A take may consist of up to about 2000 frames.

A pseudo-axis called the point of interest (POI) has been defined for special use. The value of the POI axis is the distance from the camera to the object being photographed. It is usually equal to the focus distance, but can be different if special soft focus effects are desired. The POI is defined in order to establish in 3-space the point at which to aim the camera. The location of this point is "inbetweened" in the same way as the other axes so that it will move smoothly through space. If this point does not move through space, then the camera will remain trained on this point (moving camera pan and tilt) even though the camera stand and model stand may be moving through a complicated sequence.

The ACES camera will film or rehearse a take in two modes: continuous motion or stop motion. Under continuous motion the film transport and mechanical shutter move con-

tinuously while the camera stand and model stand also move. This is the way live action motion picture filming is done. Any objects in motion will create a blur on the film which is usually a desirable effect since it looks realistic and communicates motion to the viewer. Effects filming, however, is often done with stop motion. Because the objects being filmed have traditionally been put into position painstakingly by hand and moved by hand from frame to frame, it was necessary to shoot each frame individually with all objects at rest. It was not possible to have the objects being photographed and the camera in continuous motion during filming.

With a motion control system (using servomotors—not stepper motors) it is now possible for all elements of a scene to be in continuous smooth motion relative to one another. This is most important for scenes involving fast action. Sharp images of an object in rapid motion across the field of view causes a "strobing" effect that is distracting and undesirable.



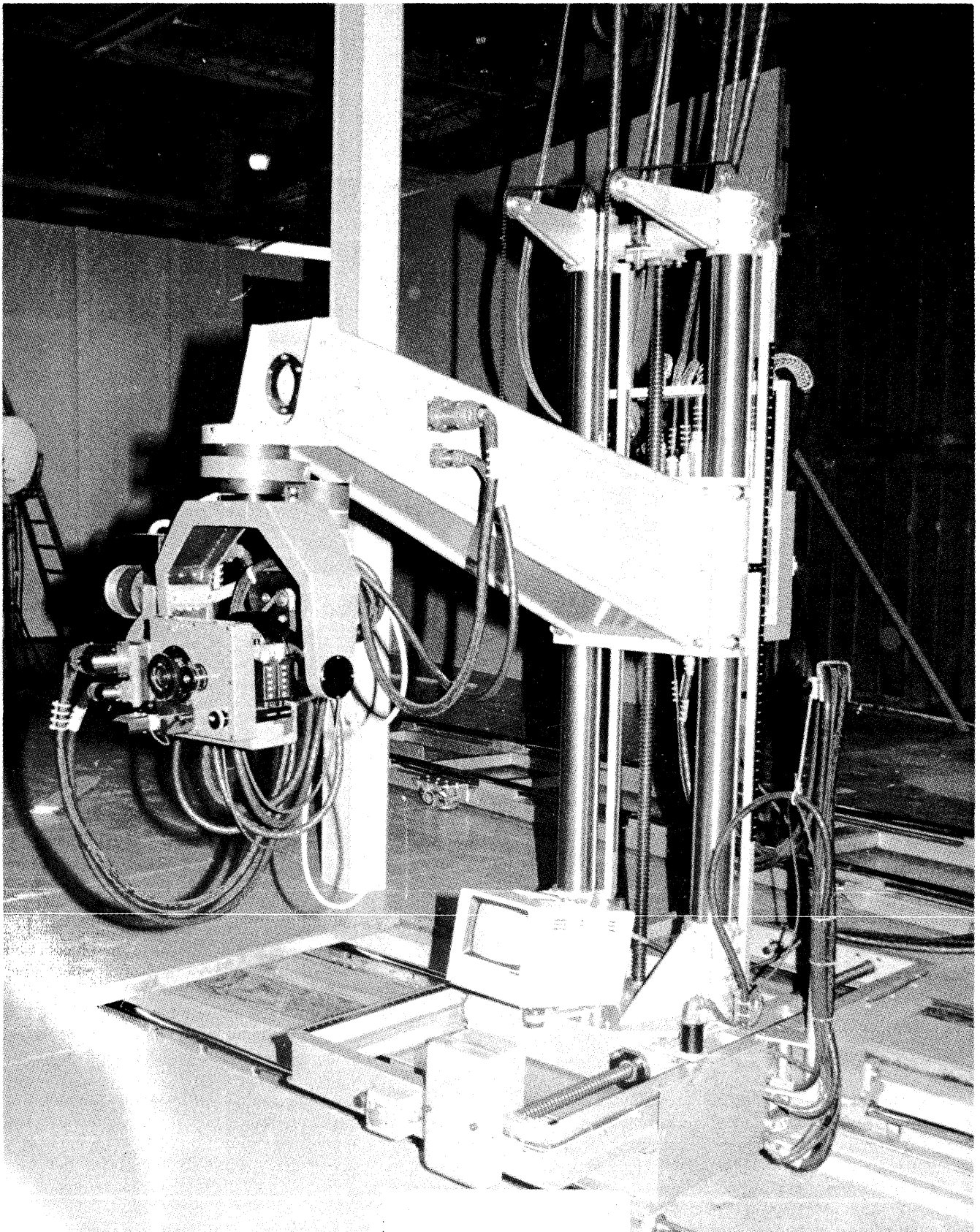


Figure 2—ACES camera stand.

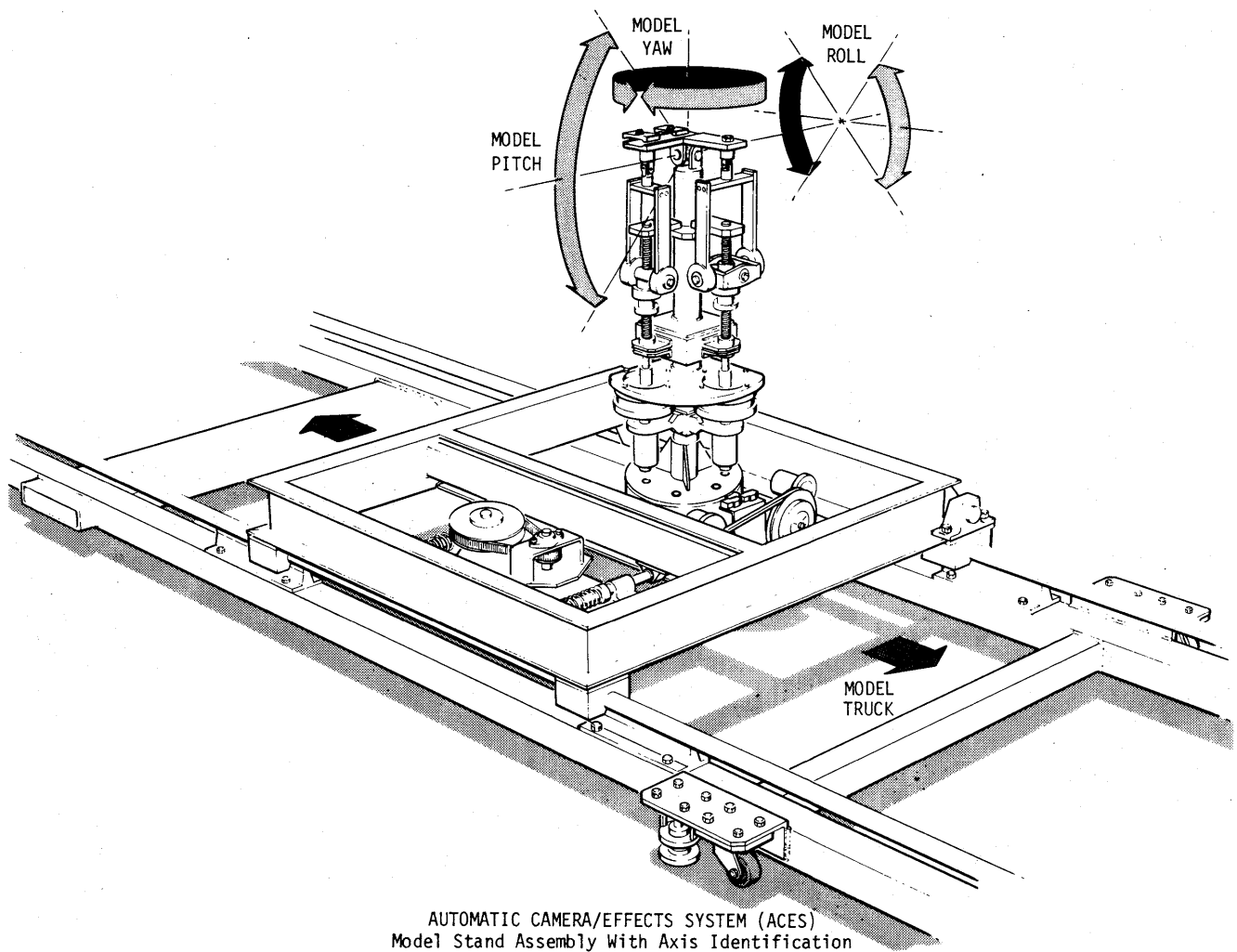


Figure 3—ACES model stand.

Another special type of continuous photography is slit-scanning. In slit-scan mode the ACES camera can create effective shutter openings of greater than 180 degrees. This increases the degree of blur and can be used to create streaking effects. Shutter openings approaching 360 degrees can be achieved by holding the shutter open for close to the entire duration of the frame and then quickly advancing from full open to full open to advance the film to the next frame. As currently implemented, the film is advanced in about 340 milliseconds. If the exposure time is 15 seconds (a typical value for miniatures photography which requires a small aperture to achieve depth of field) then the effective shutter opening is 356 degrees.

The key positions or station points are usually established by jogging the camera and model into position and viewing the scene through a boresight or video camera fixed in place of the motion picture camera. The entire movement may then be rehearsed and viewed (and recorded if desired) in black and white video. The take may then be modified if necessary by inserting new key positions or deleting or

changing old ones. Once a take has been filmed the system prevents it from being modified further. However, a take that has been filmed can be copied and the new, duplicate take can be edited.

The take data is stored permanently on floppy disks, and up to 99 takes may be put onto a single floppy. The take data consists mostly of position data but also includes such "slate" information as the type of lens used, the date and time the take was last shot, the name of the scene and production and so on.

The sequence of events that occurs in stop motion shooting begins when ACES moves all camera stand and model stand axes into position for the next frame. The computer then checks to see if a pause in shooting has been requested. The request is done either in advance via the keyboard entry or during shooting by pressing a pause button at the control console. The programmed pauses may be used, for example, to adjust set pieces through which the camera is supposed to be moving as they pass in and out of camera range or to maneuver a blue screen backing as the camera pans around.

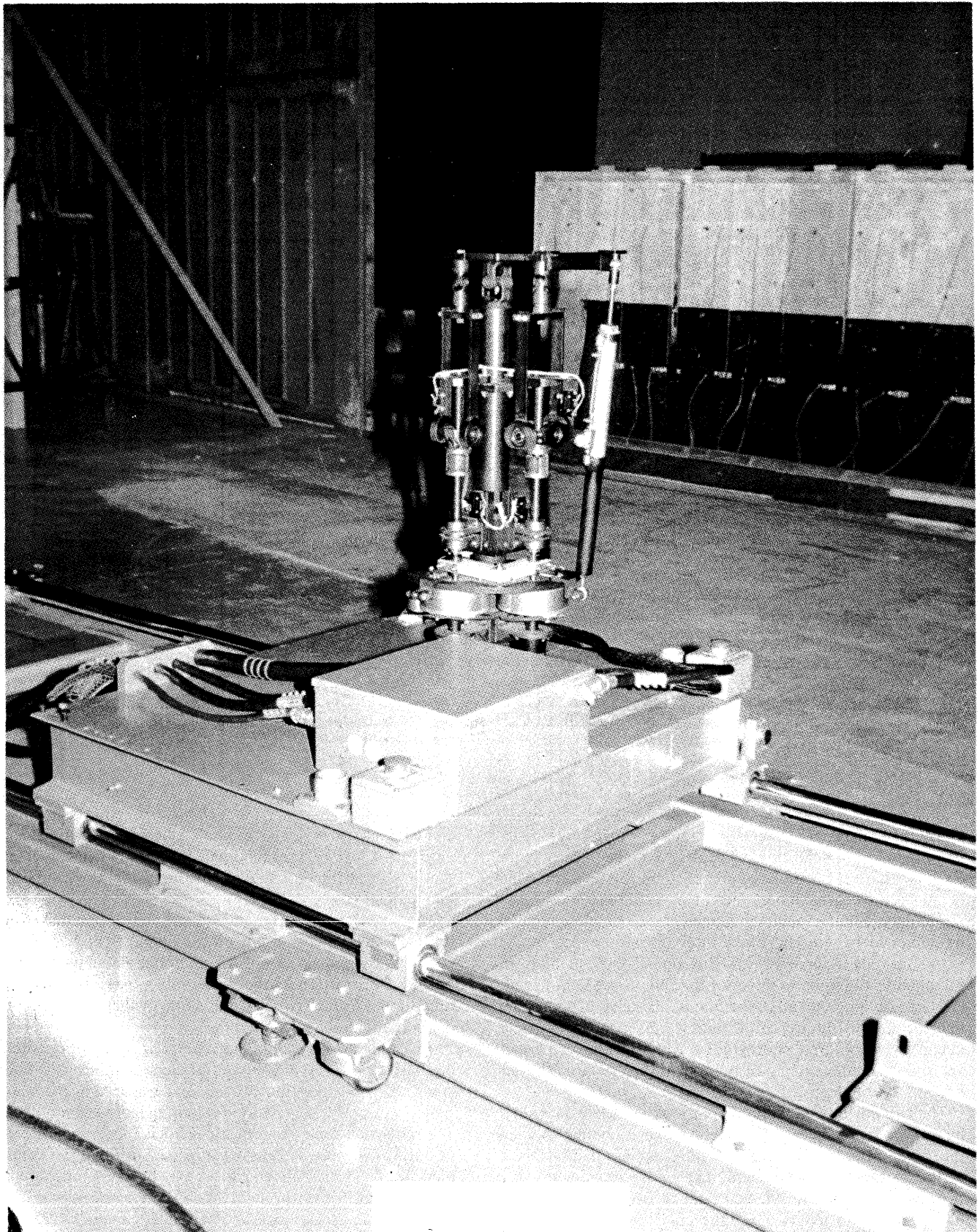


Figure 4—ACES model stand.

## ACES BLOCK DIAGRAM

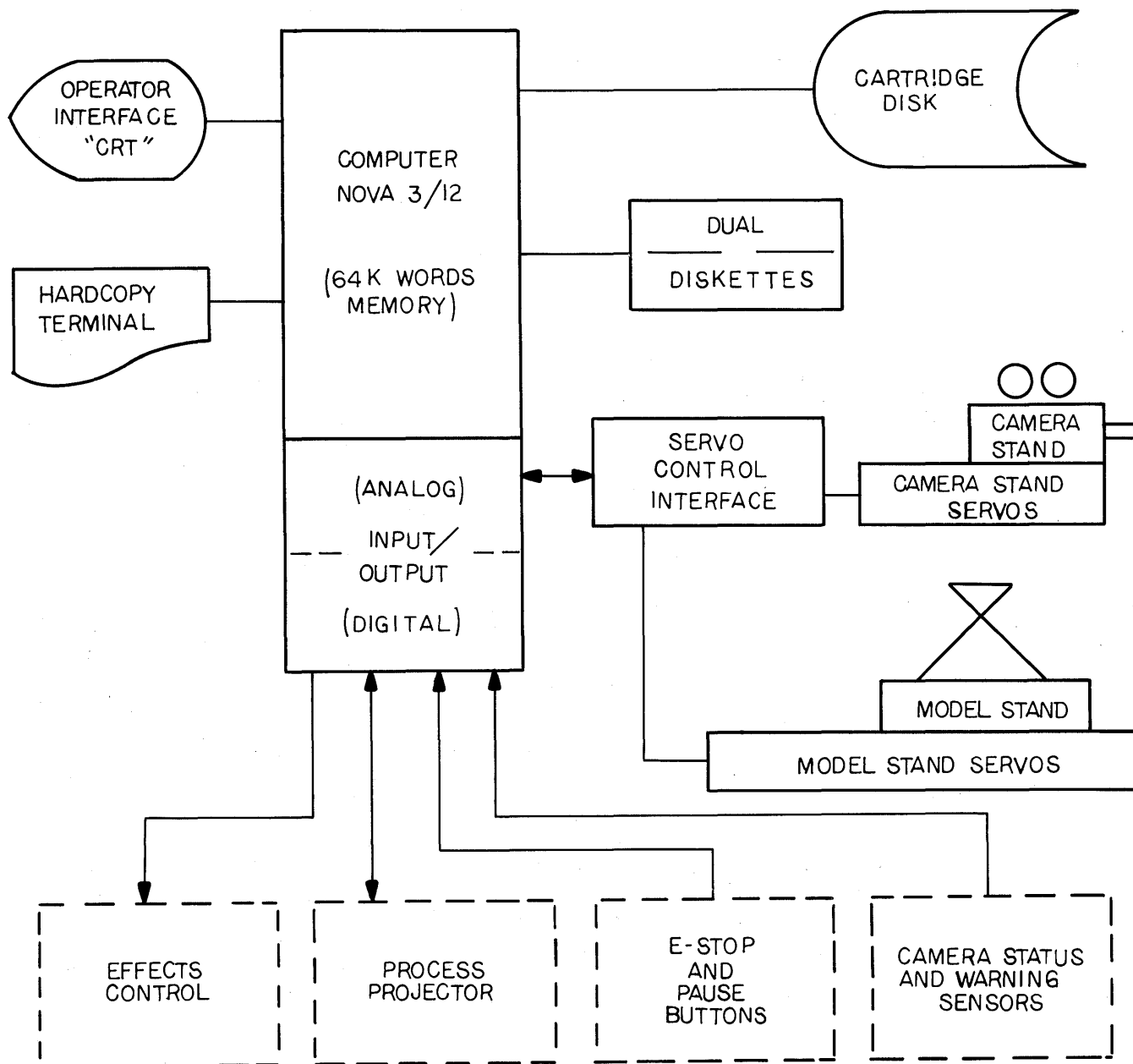


Figure 5—ACES block diagram.

The pause function is also useful for handling unanticipated events so that filming need not be aborted. If there are no pauses requested, ACES will proceed to produce any requested digital outputs. These may be used for example to advance the process projector or control other special pur-

pose devices. The system then waits for mechanical jitter to settle down. The settling time is an input variable, since the amount of jitter in the system can vary widely depending on how models and other devices are mounted on the equipment. After the settling time has passed, the transport/shut-

```

1 70:30 1:11 POSITION DATA FOR TAKE 13 (MK 13 ) ON FLOPPY 23
LAST SHOT ON 1/21/80 AT 16:52 EXPOSURE = 4.000 SETTling TIME = .00
CUTBACK TO 0 DISSOLVE FROM 0 TO 0 E-STOP AT 0
PAUSE AT 0 0 0 0 0 0 0 0 0 0
FRAME TRUCK E/W N/S PAN TILT ROLL FOCUS POI M. TRUCK M. YAW M. PITCH M. ROLL
1 1 210.00 47.00 54.50 .00 .00 270.00 137.00 137.00 4.26 2.48 -5.04 2.31
2 161 210.00 47.00 54.50 .00 .00 630.00 137.00 137.00 4.26 2.48 -5.04 2.31

```

Figure 6—Key positions for a sample take.

ter is operated to expose the film for the requested exposure time.

The execution of a continuous shot is very simple. The axes are moved smoothly and continuously through the entire take. While shooting in continuous mode, position updates are transmitted to the servo controllers every 40 milliseconds. Since position data are computed only for every frame and a frame will usually require several seconds to execute, it is necessary to compute on-the-fly position updates 25 times per second. ACES performs linear interpolation between even frame increments.

If at any time something happens to ruin the take or if the equipment malfunctions any one of several emergency stop buttons may be depressed to disable the servo controllers and let the equipment coast to a halt. The ACES software detects the E-stop condition and reverts to the command mode after displaying a warning notice. To test for correct program operation a watchdog timer is built into the servo control circuitry. If the timer is not reset by the ACES software at least every second, the E-stop comes on automatically.

The E-stop buttons are enabled at all times and may be used during stop-motion or continuous shooting or when the equipment is being moved via the jogging controls. Other sensors detect the occurrence of a break in the film, check the status of the mechanical and electronic shutters (open or closed), receive the ready signal from the process projector, and perform other similar functions.

## CONCLUSION

The greatest requirement for a system to be of value to the motion picture industry is flexibility. ACES is designed for flexibility. The boom may be reoriented in any one of four directions. The yoke mounting to the boom may also be rotated so that the yoke can be extended below to either side of or above the boom. The yoke may even be completely detached and located away from the camera stand, and in this configuration the camera stand can be used as another model stand. All of these things not only can be done but were done at some time or other during the shooting of "The Black Hole."

The same flexibility is required of the software. It was occasionally necessary to introduce a temporary patch into the software in order to achieve some particular effect. The pause provision, a variable settling time, and the slit-scan shutter, among others were first introduced in this way.

Later, they were incorporated as available options. As more options were built in, the need for programmer intervention declined, but it never ceased entirely nor is it likely to. There will probably always be an occasional requirement for some new twist.

Electronic motion control systems can most efficiently and effectively be used in conjunction with a computer. The computer interface enables the operator to think and communicate in the terms with which he is most familiar, and at the same time the availability of built-in modes of operation insures that the motion control system is correctly operated each time to produce the desired effect.

Electronic motion control is the first major innovation in the motion picture industry in a long time. The effectiveness and desirability of computer-aided techniques in film making is now firmly established, but there remains much yet to be done. Most of the tools used in the motion picture industry date back to the 1930's or earlier. They do the job, but at the expense of a large expenditure of time and specially skilled labor. The industry is fertile ground for computerization to increase productivity, lower costs, and increase both the quantity and quality of the product. At Walt Disney Productions consideration is now being given to introducing computer control or computer assistance to the cartoon cranes, the Multiplane cartoon camera, and the process lab. Given spiraling production costs and intense competition for box office receipts, the spread of computerized techniques in film making is inevitable.

## GLOSSARY

**Boresight**—attached to the ACES camera stand in place of the movie or video camera. It provides a check of the camera's point of view.

**Blue screen**—serves as a background for shots where the background is to be replaced with an image supplied from another source. For example, there may be live actors in the foreground superimposed on a background shot of a volcanic eruption, or there may be spaceship miniatures in the foreground with an image of stars and planets in the background.

**Capping**—preventing light from entering the camera box to keep the film from being exposed, especially while the film is being advanced or rewound.

**Held art**—still photograph, painting, or projected image which is given apparent motion by photographing it with a moving camera.

**Inbetweening**—in animation the process of filling in the action by creating intermediate drawings for the frames between key drawings. In ACES, interpolating positions for the frames between key positions.

**Jogging controls**—push button controls that move a selected axis a certain increment each time the button is depressed. Holding the repeat key down causes a continuous movement. There is a fast jog rate and a slow jog rate for each axis.

**Key position**—a particular location and orientation of the ACES camera stand that is required to occur at a specified frame number during a take. There may be up to 20 of them for a take of up to 2000 frames long.

**Matte**—an opaque silhouette intended to mask out a corresponding area on a film frame. The masked out area will then be replaced with an image from another source (such as a matte painting). The matte must correspond exactly to the image to be superimposed.

**Process projector**—projects an image into the scene being photographed. The ACES process projector was used, for example, to project an image of people's shadows onto the side of a spaceship model.

**Scene planning**—preparing a movement of objects and/or camera in advance by computing the desired positions at each frame based on some criterion. The criterion may be, for example, to achieve a "ratioed move" where the rate of the camera toward the object decreases with decreasing distance so that the object seems to grow in size at a constant (rather than accelerating) rate.

**Second unit**—the film crew responsible for photographing special effects and stunts. The first unit films the principal actors and standard live action.

**Shutter and film transport control**—the ACES transport mechanism holds the film in place (with pins inserted into the sprocket holes) while the film is being exposed. After it is exposed the transport pulls the film down to the next

frame, which is in turn held in place to be exposed. The (mechanical) shutter is a thin half-disk which rotates before the film plane so as to block light transmission. The film is held during the other half of the shutter's rotation. The shutter opening is usually 180 degrees but it can be adjusted manually to any value between 0 and 180 degrees. The time it takes to rotate the mechanical shutter once (and the time it takes to expose one frame and advance to the next) is called the turnover time or frame time. The exposure time is one-half the turnover time for a 180 degree shutter, one-quarter the turnover time for a 90 degree shutter, and so on. The shutter can also be operated at variable speeds. The shutter can be opened, held open for a given exposure time and then rapidly closed again. The shutter is operated in this way during stop motion shooting.

**Slate information**—the take number, scene number, director's name, cameraman's name, and other such information that appears on the slate board photographed before each take, and which uniquely identifies the take.

**Slit-scanning**—photographing with a moving camera or moving objects with the shutter held open. The effect is to create a streak across the frame for moving objects imparting an impression of great speed.

**Station point**—a key position.

**Stop motion**—photographing still objects which move between frames only while the shutter is closed. At relatively slow speeds stop motion is indistinguishable from continuous motion. At high speeds the objects create a strobing effect.

**Strobing**—when an object moving in relatively large increments between frames seems to jump rather than make a smooth move between frames.

**Take**—a continuous strip of film representing an uninterrupted sequence of action from one viewpoint.



# Automated computer controlled editing sound system (access)

by WILLIAM R. DEITRICK

*Mini-Micro Systems, Inc.  
Anaheim, California*

## HARDWARE

### *General system description*

The ACCESS hardware is comprised of eight 200 megabyte moving head, removable disk pack disk drives. There are two microcomputers, a disk drive controller, two auxiliary memory banks, three sound data channels and a two channel independent high speed memory bus (DMA) which interconnects all of the preceding devices. There are various peripheral controllers for interfacing and controlling external equipment such as video tape record/playback units, magnetic tape recorder/players, sound amplifiers, level monitors, and SMPTE code conversion units. The CRT terminal (with keyboard), a 300 line per minute printer, video monitor, speakers, and the operations console which contains, switches, indicators, and sound modification controls are all located in the operations room which is about 60 feet from the computer room where all the other equipment is located. The computer electronics hardware is mechanized on two sided printed circuit cards (PCB) approximately 5 × 8 inches, with wire-wrapped integrated circuit sockets, and interface connectors on both ends. The PCBs plug into standard 7 inch high, 19 inch wide card cages with combination wire-wrapped and printed circuit backplanes. Ribbon cables on the back edge of the PCBs interface between card cages and to external peripheral equipment. The card cages, disk drive controller, and power supplies are mounted in a standard cabinet 84 in H × 19 in W × 24 in D.

### *Major component description*

The master computer (MACPU) contain a microprocessor, CRT terminal interface, 36 kilobytes (KB) of memory, disk controller (DCU) interface, arithmetic/logic unit, interrupt logic, printer interface, a direct control interface to the monitor computer (MOCPU), and the master DMA control logic. The MACPU performs most of the data processing load, handles interrupts, issues control tables and checks status of the DCU and DMA. As the name implies the MACPU is the master of the system and essentially controls

the entire system. The CRT terminal and the printer interface directly to the MACPU. The DMA consists of two independent 8 bit data channels that allow simultaneous data transfers between any two pairs of devices on the DMA. Data transfer rates vary with the fastest rate being memory to memory transfers, 2 megabytes (MB) per second, and the slowest when the DCU is one of the devices, 806 KB/sec. There are six major devices interconnected on the DMA—MACPU, MOCPU, DCU, two auxiliary memory banks (AUX1, AUX2), and the sound channel DMA controller (SCHDMA) which coordinates data transfers to two sound data output channels (SCH1, SCH2) and one sound data input channel (SCH3). The MOCPU is almost identical to the MACPU except for the resident peripheral controllers. It handles all SMPTE code operations for synchronization purposes and has a floppy disk controller interface. It interfaces to all three sound channels via an independent 8 bit bidirectional data bus which is used to send/receive sound modification data, issue mode controls, monitor sound data memory balance counts, and is also used to control and monitor status of the external video tape equipment and audio tape equipment. The AUX1, AUX2 memory banks contain 28 KB each and are used primarily as buffer storage areas. The DCU is the most sophisticated device in the system. It can control up to 8 disk drives, each capable of storing 200 MB on a removable disk pack. Each pack can hold 40 minutes of digitized sound data so 4 hours and 40 minutes of sound effects (7 packs) can be on line and instantly available to the system. One drive is used as the system software drive and is also used to hold sound modification data, master library index, system maintenance and test routines, etc.. Normally only a few seconds of a sound effect are required to be stored in the library. That basic piece of sound can be modified in an infinite number of ways and then only the mod data need be saved which requires virtually no storage space by comparison. The DCU requires a minimum of intervention from the MACPU which initiates operations and monitors status. The DCU accesses its command tables and transfers data under DMA control to/from any device on the DMA except the SCHDMA. Command tables can be linked or chained which allows a series of operations to be performed automatically. The DCU will select a drive, move



its heads to a specified cylinder, search for a selected record or key, and when located perform a read or write data block operation, and then signal the MACPU that it is finished. The DCU can execute 25 commands and can provide full track buffering to ease timing constraints. The sound channels each contain 36 KB of memory configured as first in first out (FIFO) memory. SCH1,2 accept digital sound data and the analog sections read the data out and convert it into an analog voltage, send it to the output multiplexer which routes the sound voltage to external equipment such as audio amplifiers, level monitors, and tape recorders. The nominal output sample rate is 50 KHZ and the digital sound sample size is 12 bits. SCH3 is identical to SCH1,2 except for the analog section which accepts, scales, and conditions a sound voltage input from any external source or from the outputs of SCH1 and/or SCH2 through the input multiplexer. The input sound is digitized at a 50 KHZ rate into a 12 bit word and written into the FIFO where it is read out to the DMA. The operations console contains the modification controls and indicators for two sound channels (effectively an automated mixer with added capabilities). Each channel has a volume (level) control, frequency (pitch) control, six equalization controls that provide a range cut and boost from -15db to +15db over three frequency bands (each adjustable). There are also bandpass shape selection switches for each band. There are enable switches and indicators for each of the controls, system status and mode displays, and a SMPTE time code display. There are remote controls for both video tape and audio equipment.

## SOFTWARE

A detailed description of the software is beyond the scope of this paper and it is difficult to describe in a general manner and still be meaningful. In order to realize maximum efficiency and flexibility from the system the hardware was designed as general purpose as possible to permit continuing software development to expand the capabilities of the system. A very good balance has been achieved whereby hardware handles real-time system requirements such as sound data transfers, time code processing, interrupt generation, high speed 16 bit arithmetic and logical operations, etc. The DCU performs many operations while requiring very little intervention from the MACPU which allows maximum processing time. Programs are generally modular and self contained which permit modification and upgrading without impacting the entire system. The power of the DCU enables the computer to use virtual memory techniques which allow the system disk to look like program storage. Program data blocks can be moved between DCU and MACPU very rapidly, 36 KB in less than one tenth of a second and MACPU processing is suspended for less than one third of that. The variable record format capability of the DCU permits optimum program and data block size. Indexed sequential access methods (ISAM) techniques are used allowing the DCU to locate and fetch a sound (begin transfer) in less than one tenth of a second. The editor seldom has to wait on the system because of programming delays. The menu concept is

used where the editor is presented a list of functions, sounds, etc., which are displayed on the CRT. He then responds by choosing the desired item by entering a letter or number corresponding to that item. Direct questions are displayed requiring only a single entry to respond. Keyboard entries are kept to a minimum as much as possible. Entries are error checked and error messages displayed if the editor makes a mistake such as an invalid time code, specifying overlapping sounds on the same track, attempting illegal operations, etc. If auxiliary information is allowed such as descriptions of sounds or modified sounds the entry format is free form, i.e., the editor can enter anything he wants. There are no "computerese" type entries required. All information entered or generated during operation is saved by the system to eliminate redundant entering of data which permits many tasks to be performed virtually automatically. Complete recall and re-creation of previous work done at any level is kept available so changes can be made quickly no matter how long ago the original work was done. Reports are generated in various formats and printed for use by editors, sound technicians, and mixers. System software utilities consist of a modified INTEL assembler and editor for program generation, DCU format, test routines, disk pack evaluation, memory test programs for fault detection and isolation, and system diagnostics.

## OPERATION

A brief description of the major programs and their functions available to the editor will be covered although it should be noted that approximately one hour is required for a live demonstration of the general capabilities of the system.

### *Editor*

This program is used to fetch and edit a sound(s). The desired sound is selected by entering its key ID (up to 15 alpha/numeric characters) or the list program will display a screen full of sounds with their ID and description for selection. The display can be started at any point in the library and will be continually displayed until the end or until the editor selects one by entering a number associated with the sound. Previously modified sounds can be called by entering the mod sequence number and they also can be displayed on the CRT. The selected sounds can be played for listening only or for modification using the operations console. The sounds will be played upon command from the editor or if the VTR is being used for picture display when SMPTE time code agreement is found. (All picture material is recorded on 3/4 inch video tape cassette with SMPTE time code superimposed on the picture and also recorded on one of the audio tracks. If production sound was available it is recorded on the other audio track.) The editor can choose one of the following functions—play the sound as it exists in the library, play a previously modified sound, create background (continuously repeats the sound) or edit the sound. The edit function permits modifying the length of the sound by pro-

gressively delaying the beginning or shortening the end of the sound or will enable the editor to select any portion of the sound. After the sound is played he can again select one of the functions or if mods were made he can reset them or save them. If the mods are saved the system assigns a mod sequence number and after the editor enters a description it is entered into the library. Mods made using the operations console are retained and replayed so mods can be made to mods on successive passes without losing the previous mods (automated mix feature).

### *Prescript*

This program is used to create up to 19 sound tracks (10 min. each) for up to 23 picture reels (10 min. each). For example, a 1 hour show will typically consist of six 10 min. reels, each requiring seven sound tracks for a total of 42 tracks. The editor will assign the reel number, track number, start/stop time code, and the sound ID or mod number and the program will assign a chronological sequence number within the track and error check the entries. The accumulated track data is always available for display or printing. Once entered the entries can be modified, deleted, or moved to any tracks or reels very simply. This program also consolidates all data pertaining to the show being worked on—sounds, mods, timing data and stores it on one (or more) disk packs called the "show" pack. This permits an entire show to be played or laid off without the need for all the library packs (where the sounds originated from) to be on line. It can also be used as a library pack and allows an entire show to be saved in the physical space required for about five 35mm 1000 foot reels instead of the typical 42 reels required for a one hour show. Changes can be made and laid off almost instantaneously using the show pack. There is also a "show history" pack which contains all the data necessary to re-create a show pack except the basic sounds. A show pack is not normally retained after a show has completed final production. When a show pack is to be re-created the program will tell the editor which library packs need to be on line. Up to 80 show pack histories can be accommodated by one show history pack.

### *Play tracks*

This program permits the editor to play back one or two tracks (of the same reel). The editor enters the reel number, track(s) number, and if the track playback is to begin at a point other than the beginning of the track the sound sequence number is entered. Playback commences upon command from the editor (the SMPTE time code is simulated by the program) or when SMPTE time code agreement is found with the external equipment (usually the VTR). As with all programs the operation can be terminated at any time by the editor.

### *Record*

This program is similar to the Play Track program and it is used to lay off or record sound tracks to an external device (usually a single or multi-stripe magnetic tape recorder). An output multiplexer allows the tracks to be assigned to any of six output lines. The editor enters the reel number, track(s) number, the stripe assignment number, and if recording is to start at a point other than the beginning the sound sequence number is entered. Recording begins when the external equipment is interlocked (up to speed and in sync) and SMPTE time code agreement is reached.

### *Update library*

This program is used to load new sounds into the sound library. A sound can be input from any source (synchronized or not) and after being digitized it is saved temporarily on the system pack. It can then be played back and checked for quality, level, and then edited for length (usually several seconds is sufficient for most sounds). When satisfied with the sound the editor enters the key ID and description and specifies the library pack the sound will be stored on. The program checks the sound key ID for duplication and verifies that there is enough space on the specified library pack. The sound data is then transferred to that pack and the master library index is updated and the key entered into the ISAM table.

### *List*

This program enables the editor to display and/or print out numerous lists of information in a variety of formats (alphabetic or numeric order, time code sequence, etc.). Some of these are the contents of the sound library, modified sound library, showpack, show history pack, and track contents. Cue sheets with time ordered entries and 35mm feet/frames references for mixers and lay-off sheets for sound technicians are also provided.

### SUMMARY

ACCESS was installed at Neiman-Tillar Associates, 8304 Beverly Blvd., Los Angeles, in January 1977 and has been used to create sound effects tracks for many feature films and TV shows. ACCESS has enabled the editor to expand and utilize his creative abilities as well as increase production output fivefold. The instantaneous availability of sounds and the electronic editing capability results in a tremendous time savings. Software development continues to enable dialog clean-up and music editing to be performed on ACCESS. The second system is almost completed (pre-production model). ACCESS has become the biggest technical advance in the Post-Production sound editing field over the past 50 years.



# The use of computer technology in Magicam slave camera systems

by DAN SLATER, ROB KING and JOHN GALE

*Magicam, Inc.*  
Hollywood, California

## INTRODUCTION

With both the complexity of motion picture scenery and costs increasing, the motion picture industry is looking for cost effective methods to produce the desired film images. One approach is to inset actors into miniature sets. Maintaining correct perspective relationships between the scene components is critical to the illusion. Magicam has developed a unique system which automatically maintains a real time perspective match between scene components under dynamic conditions. This unit in essence simulates a 6 degree of freedom pantograph coupling 2 cameras at different scene scale factors.

## DIGITAL REPEAT PASS CAMERA

Several groups have developed digital repeat pass camera systems for miniature photography. These systems generally consist of a gimbaled camera on a boom arm which can maneuver near a miniature set or artwork cell. The camera is usually controlled by pulse motor drives under minicomputer control. Most importantly the camera motion may be repeated several times to form a composite image from scene components.

As an example a simple spacecraft scene could be built up from the following scene components: (1) star field background, (2) spacecraft body, (3) spacecraft lighting, and (4) spacecraft window detail.

Each component would be filmed separately because of differences in component scale or lighting. A fleet of spacecraft could be simulated by repeat pass filming of a single spacecraft.

Digital repeat pass camera motion is usually controlled by a disk, tape or memory playback of a predefined move. The move is normally produced by recording joystick motion or using curve fitting techniques. Several systems operate in synthetic coordinate spaces allowing the user to program motion in a more convenient reference frame.

## VIEWPOINT TRANSFORMATION BETWEEN SCENE COMPONENTS

The successful combination of scenes at different scale factors requires that a transformation of scene and lens characteristics be made. This is most easily visualized by examining what will happen if a camera-lens-scene system is magically shrunk. By invoking the law of similar triangles it can be proven that linear dimensions will change by the scale factor ratio while angles remain invariant.

1. The following linear dimensions will change by the scale factor ratio: (a) film size, (b) lens diameter, (c) lens focal length, (d) lens pupil aberrations, (e) wavelength of light, (f) translational motion of the lens entrance pupil.
2. The following angular measurements and dimensionless ratios are independent of the scale factor ratio: (a) angle of view, (b) lens F number (focal length/diameter), (c) the optical angular resolution, (d) rotation of the lens entrance pupil.

The viewpoint scaling transformation as described above is impractical to implement in the real world, particularly if the scale factor is dynamic. To be useful the scaling transformation must be modified to support an unscaled image plane. Since the property of rotational invariance requires that the angle of view shall remain equivalent between scene components, both the foreground and background lenses must have identical effective focal lengths. As a normal lens is focused the effective focal length will vary. By using a taking lens with telecentric properties (i.e., the chief rays exit parallel to the optical axis) the angle of view will remain constant during focus.

Since the focal length was forced to a constant value the lens diameter scaling must be by iris adjustment. Several problems arise. Image resolution in a diffraction limited lens varies as the ratio of the wavelength of light to the lens entrance pupil diameter. Since it is not practical to scale the wavelength of light, image resolution will suffer due to diffraction limiting. Special optical techniques beyond the scope of this paper are required to solve this problem.

## SLAVE CAMERA PHOTOGRAPHY

Magicam currently is using an analog based slaved camera system (ASC) operating under the constraints of the viewpoint scaling transformation. The ASC consists of a foreground camera unit, control unit and a background camera gantry.

The foreground camera photographs the full scale scene component, usually actors, against a chroma-key blue screen. The foreground hardware consists of a standard broadcast color TV camera mounted on an instrumented dolly providing 5 degrees of freedom. The position of the camera viewpoint is determined by a small analog computer reading the various dolly sensors.

The dolly steers in the crab mode which corresponds to all 4 wheels pointing in the same direction simultaneously. This results in the dolly heading remaining constant independent of the direction of travel. As a first order approximation the wheel rotation rate and steering angle correspond to the magnitude and direction of the horizontal velocity vector. The velocity is sensed by a wheel tachometer and resolved into cartesian coordinates by a sin-cos pot measuring the steering angle. The foreground lens viewpoint offset is computed and added to the position data prior to the scaling operation.

The background unit consists of a probe lens camera system mounted on a 6 degree of freedom gantry. The probe lens is similar to a submarine periscope. This allows maximum maneuverability in the miniature scene since in theory the background camera physical dimensions and height should be reduced by the scale factor ratio. The probe lens viewpoint height can under certain conditions approach the miniature floor level.

The system uses DC servos to position the various axes. Generally all servos must be fast, smooth and have low dynamic errors. The rotational axes typically have noise levels below 10 arcseconds and a velocity lag error constant (Kv) greater than 500.

The horizontal axes are driven by velocity servos which integrate the foreground velocity command signals. To maintain an acceptable integration error tolerance it is necessary to use both a high loop gain and servo nonlinearity modeling. Since the servos are operating as integrators the constant of integration must be defined. This is most easily established by measuring the distance in both the foreground and background to targets at equivalent scaled distances.

The vertical axis uses a leadscrew drive to physically raise and lower the complete camera and probe lens assembly. Even though this axis operates as a positional servo it is necessary to initialize it by adding in an arbitrary offset. This is necessary to set the correct height ratio between the foreground and background scenes. Note that the background floor height is at an arbitrary level relative to the foreground.

The lens yaw is accomplished by rotating the complete camera and probe assembly with a direct drive DC torquer. The lens optical path goes through the center of the torque motor. Lens tilt is accomplished by servoing a mirror mounted at the probe tip.

## ASC DEFICIENCIES

The ASC, while successfully in use today in television productions, is not adequate for feature film production. The ASC lacks the level of precision required for large screen work and does not have move playback capabilities.

The precision of the perspective match is limited primarily by three error sources.

### *Lens aberrations*

The ASC optics have an excessive mismatch between entrance pupil aberrations of the foreground and background lenses. The entrance pupil aberrations do not degrade the image but instead cause an angle dependent scene misregistration. The entrance pupil aberrations correspond to an angular distortion of the camera viewpoint.

### *Attitude determination errors*

The ASC foreground attitude is referenced to the floor which generally is not flat.

### *Roll errors*

Camera roll is not sensed and therefore is not reproduced in the background unit.

## MAGICAM MCS

The Magicam modular camera system (MCS) is a highly modularized digitally based camera system. The MCS is essentially an integration of several concepts including a Magicam analog slave camera system and digital repeat pass camera technology. As a combination of several techniques it provides more than the sum of the respective capabilities.

The MCS uses 35MM motion picture cameras to photograph the scene components, requiring significantly higher accuracies than the ASC.

The MCS consists of the following modules: (1) foreground position measuring system (FPMS), (2) foreground camera (FCAM), (3) digital computational system (DCS), (4) background gantry assembly (BGA), (5) background camera (BCAM), and (6) background abort sensing and implementation system (BASIS).

## FOREGROUND POSITION MEASURING SYSTEM (FPMS)

Because of problems related to a floor based attitude reference used in a previous unit, the MCS attitude data is derived from inertial sensors referenced to the g vector and artificial stars in the stage area. The foreground position is determined by an inertially smoothed microwave transponder. A CCD startracker boresighted through the camera op-

tical system provides a primary alignment referenced to the camera viewpoint.

The strapdown inertial measurement unit (IMU) provides body relative yaw, pitch and roll rate signals which are integrated to form an earth relative viewpoint attitude. The attitude matrix is used to resolve the IMU accelerometer outputs into the earth reference frame. The acceleration components are integrated twice to form a viewpoint position.

To maintain adequate long term accuracy it is necessary to estimate the hardware errors. This is accomplished by combining the IMU outputs with ranging data derived from a microwave transponder in an extended Kalman filter.

The Kalman filter is configured as a partial feedback filter operating in the error state space. The error state vector includes 3 gyro bias, 3 accelerometer bias, 3 velocity error and 3 position error state elements. To maintain acceptable performance, approximately 1 filter update per second is required.

As compared to conventional navigation systems, the FPMS operates in a significantly smaller area, with a higher sample rate and a higher translational axes measurement accuracy.

FPMS performance design requirements:

	Angular	Translational	
Range limit .....	Continuous	50. meters	
Rate limit .....	540. deg/sec	100. cm/sec	
Acceleration .....	5400. deg/s/s	50. cm/s/s	
Resolution .....	10. arcsec	2. mm	(1 sigma)
Accuracy .....	60. arcsec	3. mm	(1 sigma)
Sample rate .....	96. Hz	96. Hz	

## CCD STAR TRACKER

The FPMS star tracker provides two separate functions. As a startracker the foreground CCD sensor detects artificial stars in the stage area. Alternately the CCD video output may be combined with the video from a similar background CCD sensor to form a preview composite image.

The FPMS must be aligned relative to the camera viewpoint. The camera viewpoint corresponds to the camera lens entrance pupil. By placing a series of artificial stars in the stage area, the lens pupil location can be determined by star tracking. A CCD imaging device is coupled to the film camera optical system by a beam splitter and a format conversion relay lens. The relay is required because a full frame image is needed for the composite scene preview function.

The CCD sensor samples the field of view as a discrete  $512 \times 320$  pixel array, corresponding to approximately .1 degree per pixel in the object space. By computer processing it is possible to resolve star positions to a small fraction of a pixel. Tests with the RCA CCD sensor indicate star positions can be resolved to 20 arcseconds. The pixel interpolation is accomplished by first suppressing images other than stars and then determining the energy centroid of a defocused star. Spurious images are suppressed by subtracting a star off image from a star on image.

## DIGITAL COMPUTATIONAL SYSTEM

The digital computational system consists of two general purpose mini-computers with disk and 9 track tape storage, an interactive graphics terminal and the associated software required to operate the system. One computer handles the FPMS while the other handles system control and the background hardware.

## MCS CONTROL PROGRAM STRUCTURE

The MCS control program is heavily based on state space techniques as developed in modern control theory. The system state is defined as a minimum set of variables called a state vector. The state variables are those variables which determine the future behavior of a system when the present state of the system and the excitation signals are known.

The dynamic behavior of the system is determined by the state transition process where the new system state is formed as a function of the current state and any deterministic forcing functions. In the MCS the state transition is formulated in a discrete time manner.

By formulating a state vector at a basic level, program modularity is enhanced since intertask communications are handled naturally by the state vector.

The core of the MCS program consists of the following program modules.

### *State vector compiler*

The state vector compiler allocates and produces named references to an area in core where the state vector resides.

### *State initialization*

The state initialization program copies the state vector to and from disk in core image form. This routine is used both to initialize the system and for graphics oriented nested control structures.

### *State vector editor*

The state vector editor provides a human interface to the values in the state vector. Values may be both displayed and modified.

### *Synthetic servo handler*

The synthetic servo handler transforms the synthetic axes motion state to real axes motion, provides predictive (lead) motion estimation to compensate for servo desampler lag, and provides error checking.

### *Foreground interface*

This routine is an interface to the foreground viewpoint position estimate computer.

### *State transition program*

The state transition program transforms the current system state to a new system state 1 sample interval into the future. The new state is a non-linear function of the current state and deterministic control functions.

### *State transition program editor/compiler*

This routine is a special purpose editor which allows the user to display and modify the state transition program using a film oriented language. Provisions are included to partially invert the state transition program for reverse time motion.

The MCS program is coded in the FORTH language. FORTH is a unique and powerful programming technique providing the required speed and language extensibility necessary for this application. In essence FORTH simulates a set of extensible virtual stack oriented processors which are programmed in a virtual macro assembly language.

## BACKGROUND GANTRY

The MCS uses the same background gantry as the ASC. The gantry has been modified to accept a new probe lens and motion picture camera. In addition a CCD camera has been boresighted on the optical axis to allow verification of scene component registration.

The gantry is under the control of a digital closed loop servo system which was specifically designed to operate in a sampled data environment. The servo system utilizes a phase coherent desampler with a wideband feedforward error estimator to reduce scene component tracking errors

to an acceptable level. For repeat pass motion playback the desampler time scale factor may be varied under computer control.

## BACKGROUND ABORT SENSING AND IMPLEMENTATION SYSTEM (BASIS)

Because of the relatively high cost and fragile structure of the MCS probe lens and miniature sets, positive protection is required. The Magicam MCS incorporates both background fault sensing and abort implementation hardware to protect both the probe lens and miniature. The BASIS operates in an autonomous manner determining the fault status or risk present. When the risk exceeds a threshold the computer is interrupted for an orderly system abort. If the risk level exceeds a higher threshold the BASIS will assume that the computer has lost control of the MCS and will therefore execute a priority abort independently of the computer. For diagnostic purposes the BASIS maintains a fault event queue.

## CONCLUSIONS

The basic design of the Magicam MCS slave camera has been described with particular emphasis on the computational requirements. The system software uses a state space problem description to enhance the program modularity, structure, and adaptability.

## REFERENCES

1. Dykstra, J., "Minature and Mechanical Effects for *Star Wars*," *American Cinematographer*, pg. 702, July 1977.
2. Kingslake, R., *Lenses in Photography*, Garden City Books, Garden City, New York, 1951.
3. Slater, D., et al., "Image Synthesis with Servoed Cameras," *Simulators & Simulation SPIE*, Vol. 59, 1975.
4. Dorf, R. C., *Modern Control Systems*, Addison-Wesley, Reading, Mass., 1967.
5. Forth Inc. "Using FORTH," Forth Inc., Manhattan Beach, Calif., 1979.

## Telecommunications

The telecommunications area contains three varied sessions. These sessions cover the entire spectrum of telecommunications from terminal interfaces to data management in a distributed environment.

The session chaired by Dave Peters will provide an overview of modems, multiplexors, and system interconnection design considerations. The session will concentrate on interface and communication network design fundamentals.

The session chaired by Doug Theis will describe the state-of-the-art in minicomputer networks. It will cover the history, motivation, structure, capabilities, and future trends associated with network architectures built around minicomputers.

The last session is chaired by Dr. Robert Hall and will discuss data management in networks. It will consider the problem, analytic techniques, implementation approaches, and potential of data base machines in network structures.



**Kenneth J. Thurber**  
*Area Director*





# Distributed network and multiprocessing minicomputer state-of-the-art capabilities

by DOUGLAS J. THEIS

*The Aerospace Corporation*  
El Segundo, California

Today's minicomputer and midicomputer state-of-the-art provides two basic types of capabilities for users to evaluate for their applications. The two categories are:

- Distributed Networking Systems
- Multiprocessing Architectures with High Speed Buses, Shared Main Memory and Shared Disk Systems

A distributed network system (e.g., Digital Equipment Corporation's DECNET, Hewlett Packard's DSN, Modcomp's MAXNET) offers standard data communications interfaces with line protocols over both telephone lines and hardwired cables. On the other hand, multiprocessor approaches offer several interconnection schemes. An interprocessor bus (e.g., Data General's Multiprocessor Communication Adapter and Digital Equipment's Parallel Communication Link) can provide high data transfer rates among processors, with either custom user-developed software or a vendor developed real-time operating system. This bus interconnect approach is also evolving into configuration implementations with a significant amount of fail-soft or graceful degradation capability (e.g., Tandem's minicomputer and BTT's midicomputer system) and a turn key operating system that fully supports multiprocessing.

Practical multiprocessing is also achieved by sharing memory of some sort so that many CPUs can access one common memory. Shared main memory provides the necessary multiprocessed capability so that two or more CPUs can access one common memory. This means that special instructions and software are required for a well disciplined method of resolving contention and simultaneous memory request conflicts. The shared memory multiprocessing can also be achieved with a shared disk unit, where two or more computers have access to one common disk. Shared disk provides more memory capacity, but the time to read or write data is longer. Most major minicomputer and midicomputer manufacturers offer shared disk hardware capability. Table I gives a status on minicomputer and midicomputer vendors which support both distributed networking and multiprocessing capabilities.

## DISTRIBUTED NETWORKING MINI/MIDICOMPUTER SYSTEMS

Distributed network processing is definitely upon us and being used in many applications, but the availability of off-the-shelf, turn key distributed networking systems has been limited. Although the necessary hardware, such as data communication controllers and interfaces, has been available, the constraining force has been the development of the software. The total objective is to make the hardware links and message/communication protocols transparent to the user; as a result each vendor has had to make many design decisions about the network control methods and access methods. All this is needed in order for the user to have higher-order level language statements for network operation using just logical references, which in turn do the remote operations with other nodes in the distributed network. Even now only some of the major vendors offer substantial support, but just about all the others are developing this capability to catch up with the obvious trend.

Each manufacturer has started out offering partial capabilities because it is really not practical to develop everything one could think of as desirable. For example, Hewlett Packard's first network system, which interconnected distributed minicomputers throughout their own factory, was designed to provide peripheral sharing and central programming development services. So far each vendor has gone about providing distributed network capabilities in a somewhat different manner, yet providing means to accomplish similar types of end objectives. It is difficult to compare vendors' offerings of distributed network capabilities for the very reason that there are so many different ways to accomplish similar top-level operations.

There are at least two dimensions to a distributed network mini/midicomputer system: one dimension is the network configuration (i.e., topology of nodes) and the other is the layers within each node, from the bottom layer of the electrical interface to the top layer consisting of the network access method, which allows the user complete transparency of the network characteristics. The major topics discussed

TABLE I—Mini/midcomputer vendor support for four configuration types

Manufacturer/ Model Nos.	Type 1 Distributed Network	Type 2 Multiple CPUs Interconnected With Bus Hardware	Type 3 Shared Main Memory	Type 4 Shared Disk
BTI Computer Systems, Inc. BTI 800		X		X
Computer Autom- ation Inc. LSI 4/10, 4/30, 4/90	X			X
Control Data Corp. Cyber 18 Series, MP 60	X			X
Data General Corp. Nova and Eclipse Series	X	X		X
Digital Equipment PDP 11 Series VAX 11/780	X X	X	X	X X
General Automation 440, 460, 440DS, 460NDS, 550	X			X
Harris Computer Systems Slash 4, 6, and 7 Series			X	X
Hewlett Packard 1000 and 3000 Series	X			
IBM 8100 Series	X			
Modular Computer Modcomp II, IV, and Classic Series	X		X	X
Perkin-Elmer (formerly Interdata) 7/32, 8/32, 3220, 3240			X	X
Prime Computer F350, 450, 550, 650, and 750 Series	X			X
System Engineering Laboratories, Inc. 32/35, 32/55, 32/75			X	X
Tandem Computers T16 Series	X	X		X
Texas Instruments 960, 980, 990				X
Univac (formerly Varian) 77-400, 77-600, 7-800	X		X	X

within this distributed network section are as follows:

- Topology
- Line Protocols
- Hardware Interfaces
- Network Protocol
- User Software
- Network Software

The user learning about distributed network capabilities quickly finds many new acronyms and terminology related to this industry. Some of the more common ones are presented in Table II.

### Topology

Network configuration, or topology, refers to the interconnection pattern of nodes and links to form the network. The most commonly used configuration of networks is point-to-point, where each node has its own links to other nodes. Multidrop is the other type of configuration (sometimes

called multipoint); it is different in that two or more nodes time share a link to save on the communication link costs. Point-to-point capabilities are supported by all the Type 1 vendors presented in Table I. Multidrop was available from some vendors in 1979 and will be supported by others sometime in 1980. Many point-to-point configurations are often found in hierarchical networks where the top level is the host or central node (as in a star network) with the second or bottom level being satellite nodes collecting and processing data in a distributed fashion. Differing topologies are exemplified by Digital Equipment Corporation's DECNET which supports any point-to-point network combination, and by IBM's 8100 which emphasizes a loop or ring structure topology locally. Any distributed network topology has the usual address field or software table limits as to the maximum number of nodes, links, device status, number of paths, etc. which is part of any finite system, but these can be extended when necessary both in size and for more complex applications.

### Line protocols

The communication line protocols (i.e. link level control procedures) used over these circuits are some of the most confusing and difficult aspects in distributed network computer systems. During the 1960's remote batch applications

TABLE II—Distributed network related terminology/definitions

#### BASIC TERMS

**Node** - can be a terminal or computer termination point which has the capability to transmit and receive over communication line facilities; usually an addressable entity.

**Link** - the external circuit connection to provide a means of exchanging data via telephone, hardwired cable, or private communication facilities.

**Topology** - the interconnection of nodes and links into a network configuration for overall operations of one kind or another.

**Protocol** - a formal set of rules for specifying the format and relationships of message exchanges among communicating nodes.

#### NETWORKING CONFIGURATIONS

**Point-to-point** - a network configuration or topology between two nodes with a direct link (i.e. two nodes are connected without any intermediate nodes in between).

**Multidrop** - a communication link on which two or more nodes (computer or terminal) can share access. This type of link typically requires both polling (or a reservation scheme or contention) and addressing techniques to facilitate shared link operation.

**Loop or Ring** - a ring like network configuration where each computer node is connected to two adjacent nodes forming a closed loop.

#### NETWORK SWITCHING

**Circuit switching** - an electrical connection between two nodes (i.e. over a communication link) is established on demand for dedicated bandwidth of that circuit (i.e. link) until it is dropped.

**Message switching** - is a capability for messages to be moved over many intermediate links and nodes because the destination address is integral to the message to allow variable paths to be taken as availability permits.

**Packet Switching** - is a subset of message switching where messages are broken into packets including their own necessary address and control as to their destination. Packet switching allows efficient use of network resources because it breaks up data into a relatively small size (e.g. X.25 allows up to 256 bytes) and no long term storage is required at the various routing nodes.

**Virtual Circuits** - is a logical circuit connection between two nodes in a network which can be realized with different circuits during the transmission of a message. This service is guaranteed, sequential related operations including end-to-end flow control.

**Datagrams** - is a capability which provides link communication on non-related (may or may not be guaranteed), non-sequential operations basis primarily toward transaction oriented users where the users provide end-to-end integrity.

and terminals used character oriented protocols such as IBM's well-known BISYNC (i.e. BSC) and others. During the 1970's, the trend toward transaction-oriented systems drove the industry into the era of new protocols such as IBM's SDLC. Also DEC has its DDCMP byte protocol which is designed using a count field instead of a unique flag character as is done in the SDLC bit protocol. Thus bit stuffing protocols (e.g., IBM's SDLC) versus DEC's imbedded character protocol is a controversial subject in that there are trade-offs associated with each. Table III shows a comparison of the communication line protocol presently supported by mini/midcomputer manufactureres as well as BISYNC which is still used for reasons of compatibility with the older remote batch terminal and display terminal equipments. In Table III there is a breakdown of some specific capabilities each of the five protocols supports. There are many other factors beside these to consider in the area of link protocols, but it is beyond the scope or overview we are trying to present in this article.

One new trend that is certain to be offered by more vendors is the use of an integral microcomputer to do the line protocol functions and thus decrease the amount of code and overhead in the node distributed network software subsystem. In particular Digital Equipment Corporation offers two

types of microprocessors: (1) DMC-11 with a ROM memory implementing their DDCMP protocol and (2) KMC-11 with a RAM memory so any other protocol (e.g. IBM's) can be implemented. Note that the Signetic's protocol chip advertised for doing the DDCMP protocol does the framing function which comprises only a part of the operations for the DDCMP protocol. In the case of Hewlett Packard, their line protocol software is implemented in the node minicomputer microcode/control store memory. This off-loads part of the Hewlett Packard Distributed System Network software resident in main memory. Hewlett Packard plans to use an integral microcomputer in their next DSN upgrade. These types of approaches should improve throughput performance.

#### Hardware interfaces

Distributed network computer system products are new systems and as such high "effective user" data rate throughputs are difficult to achieve until after functional software design and operational experience has matured. The throughput inefficiencies are primarily in the software overhead features developed to make system network details

TABLE III—Line protocol features and characteristics

FEATURE	BISYNC	DDCMP	ADCCP	HDLC	SDLC
Protocol Type (i. e. Data Transparency)	Character Stuffing	Count	Bit Stuffing	Bit Stuffing	Bit Stuffing
Full Duplex	No (HD)	Yes	Yes	Yes	Yes
Serial	Yes	Yes	Yes	Yes	Yes
Parallel	No	Yes	No	No	No
Multi-Acknowledge by one ACK	No	Yes	Yes	Yes	Yes
Asynchronous Operation	No	Yes	No	No	No
Point-to-Point	Yes	Yes	Yes	Yes	Yes
Multipoint	Yes	Yes	Yes	Yes	Yes
Error Detection (CRC)	CRC-16	CRC-16	CRC-CCITT	CRC-CCITT	CRC-CCITT
Retransmit Error Recovery	Yes	Yes	Yes	Yes	Yes
Count Field Required	No	Yes	No	No	No
Protocol Overhead (single data message)	80 bits and DLE additions	80 bits	48 bits fixed and O's for bit stuffing	48 bits fixed and O's for bit stuffing	48 bits fixed and O's for bit stuffing

transparent to users. Then the more network support software that is provided to do all these nice features drives up the complexities which need to be refined, tuned and optimized to maximize throughput performance. This is easy to say but obviously requires the appropriate software/hardware trade-offs of how much complexity is really necessary and some smart design implementations to achieve the highest throughput performance possible.

The physical links between nodes are either data communication lines, with modems and adapters attached to the computer channels, or hardwired cables. At the device level, the EIA (Electronics Industries Association) RS-232-C standard is provided by all the vendors. Other standard circuit connections that are either supported or planned to be supported include the following: Electronics Industries Association EIA RS-422 and RS-423, Western Electric 301/303, MIL-STD-188C and the new Opto-Isolated fiber optics interfaces. RS-232-C provides operation up to 19,200 bits per second. Higher data rates and better performance can be accommodated by the newer EIA RS-422 (up to 10M bps) and the EIA RS-423 (up to 100K bps) circuit standards. The Western Electric 301/303, supported by several vendors such as Modcomp, permits data rates from 19,200 bits per second up to 230,400 bits per second over special leased circuits. MIL-STD-188C is very similar to RS-232-C and vendors can support it in special situations as needed for the government and the military. So far standards for opto-isolated interface have not been developed to the same level as the above electrical interface standards.

The vendors in Table I support many but not all of the commercially available data link services from voice grade telephone lines up to 9,600 bits per second, leased (private) lines up to 230.4K bits per second, DDS (Dataphone Digital Service) up to 56,000 bits per second, Bell's T1 Carrier and Satellite Carriers (e.g. Western Union's Weststar) both up to 1.544M bits per second. Presently Digital Equipment Corporation's DECNET II using their DDCMP protocol has hardware interface and software drivers to accommodate 9,600, 19.2K, 250K, and 1M bits per second for point-to-point hook-ups using the appropriate modems or special cable interfaces. For multidrop arrangements, Digital Equipment Corporation provides both the low speed 9600/4800 (outbound/inbound) bits per second with modems and a high speed, 512K bits per second, synchronous, full duplex capability over cables. Hewlett Packard's DSN in point-to-point networks accommodates up to 56,000 bits per second with modems and up to 480,000 bits per second over cables. DSN supports both the low speed 9600/4800 (outbound/inbound) multidrop configuration as well as a 19,200 bits per second multidrop capability. Modcomp's MAXNET supports point-to-point configurations up to 19,200 bits per second with RS232-C compatible modems, 40.8K with WE 301 modems, and 19.2K, 50K, 230.4K with WE 303 modems. Without modems in a point-to-point configuration, MAXNET supports either 19,530 bits per second or 1.25M bits per second using cables and internal clock. MAXNET supports multidrop using RS232-C modems capable of multidrop up to a maximum of 9,600 bits per second outbound and 4,800 bits per second inbound. Modcomp also supports 250K

and 500K bits per second multidrop for up to 5,000 feet with a limit of 8 drops (distance is dependent on speed and number of drops).

### *Network protocol*

There is a network standard and that is the CCITT (International Consultative Committee for Telegraphy and Telephony) X.25 network access protocol. It is a framework within which to build a system that allows vendors to implement their own efficient software designs as well as inclusion of many additional or optional features. Several vendors have already begun to implement a compatible version for their own distributed network systems to provide the necessary control functions, message formats and other capabilities necessary to implement the compatible node to node operations so that the user can do normal network access operations with other X.25 networks. The X.25 Network Access Protocol was developed by CCITT Study Group VII in July 1976 (revised April 1977) and is supported by Telenet, Datapac (Canada), Euronet (Europe), and others. The X.25 protocol consists of three access levels:

1. Level 1 defines the physical and line interfaces used (i.e., RS-232-C).
2. Level 2 defines the link level procedures in terms of message formats and sequencing (i.e., HDLC).
3. Level 3 defines the packet format used to establish and exchange data (i.e., virtual circuit).

For example, Digital Equipment has under development an X.25 compatible interface for its own DECNET so user tasks and DECNET utilities can communicate with a remote computer node over any public packet switching network using X.25 protocol. Two program interfaces are needed to provide a common basis for call, set-up, call disconnect and exchange of data between DECNET machines and other types over a X.25 network. Hewlett Packard plans to interface to these public X.25 networks and Modcomp also plans to have X.25 software interface compatibility in the not too distant future. Table IV provides information on each vendor's network access protocol/layers of software to show the overall framework and its related terminology.

At least one manufacturer has announced a chip (i.e., Western Digital's 2501) that is specifically designed to replace the software required for the first two levels (physical and link access) of the X.25 packet switching protocol. The VLSI chip has approximately 30,000 gates. Maximum data rates of up to 1.3M bits per second are planned to be accommodated by this chip. When level three (network control) is firmly specified, then it will also be put on a chip. In addition, this chip design provides link and loop back testing and has a programmable address field to allow terminal intercommunications. Prior to the X.25 being on one or a few chips, it was too expensive to make intelligent terminals as nodes in a public network and now that trend is changing. Note, however, that terminal networks is not the subject of this article.

TABLE IV—Manufacturer's network software layer/protocol structure

Manufacturer	Network Architecture Acronym/Designation	Mini/Midi Computer Models Supported	User Program Interface	Network Link Level	Message Protocol (e.g. SDLC)	Link Hardware Interface (e.g. RS 232-C)
Computer Automation Inc.	SyFA (System For Access) VN (Virtual Network)	LSI4/10, 4/30, 4/90	Not Supported	Emulates IBM 3270, 3780, others	Bisync	RS 232-C
Control Data Corporation	NAM-18 (Network Access Method)	Cyber 18 Series	Under development	X.25 Compatible under development	Mode 4, CDCCP (ADCCP, HDLC, and SDLC)	RS 232-C
Data General Corp.	None, not developed	Nova's and Eclipses	Not Supported	X.25 Supported	HDLC	RS 232-C
Digital Equipment Corporation	DNA (Digital Network Architecture)	PDP 11 Series VAX 11/780	Vendor offers several: DAP, ATS, User	Vendor calls Network Service Protocol (NSP)	DDCMP	RS 232-C WE301/303 CCITT V. 35
General Automation Inc.	AUTONET	Models 220, 440, 460DS, 460 NDS, and 550	Under Development	X.25 Compatible under Development	ADCCP, SDLC	RS 232-C
Hewlett Packard Inc.	DSN (Distributed System Network)	1000 Series 3000 Series	Vendor calls Network Access Method	Vendor calls Network Manager DS/1000 DS/3000	ADCCP, HDLC	RS 232-C WE 301/303 CCITT V. 35
IBM	SNA (System Network Architecture)	8100 Series	Function Management	Transmission Control & Path Control (2 layers)	SDLC	RS 232-C
Modcomp Inc.	MAXNET	Modcomp II, IV and Classic Series	Vendor calls REX	Vendor unique	ADCCP, HDLC, SDLC	RS 232-C WE 301/303 CCITT V. 35
Prime Computer Inc.	PRIMENET	350, 400, 450, 500, 550, 650, and 750	Inter Program Communications Facility (IPCF), File Access Manager (FAM)	Packet Network Interface (PNI) X.25 Compatible	HDLC	RS 232-C
Tandem Computers	GUARDIAN/EXPAND	Tandem 16 Series	Vendor Unique	X.25 Compatible under development	HDLC	RS 232-C
Univac (formerly Varian)	DCA (Distributed Communications Architecture)	77-400, 77-600, 77-800	GRAM (Global Resource Access Module)	Vendor calls Transport Network System	HDLC handle HDLC and SDLC as sub-sets	RS-232-C

The network control software module layer (typically level 3) below the user's level takes their requests and decodes them as such into link and node assignments. These assignments are allocated at system generation time so the necessary reformatting and control is already established to carry out the desired operation. There is no standard because each vendor defines and implements it to suit himself. Some vendors have more layers in their software network structures than others. This software does the routing, sets the control fields according to prescribed methods, and performs any other functions which are necessary to provide desired operations at another node. The network control software can be designed to provide alternate routing as well as other items related to network transfers. When there are more than two levels of nodes (i.e. intermediate nodes between sending and receiving nodes), some degree of routing capability is needed. There are many and varying degrees of capability one can implement for network routing. One basic capability

is to provide automatic rerouting when a node fails, but it may require the user to reinitiate the job. The objective is to make rerouting or changing link paths transparent to the user. Dual path capability between nodes with some degree of line load sharing to maximize link utilization is an additional complexity to provide in network routing. Vendors start out offering some basic routing capabilities and then evolve into more complex routing capabilities as time goes on. It is well to remember that routing capabilities can vary from some basic features all the way up to some very sophisticated ones.

### User software

At the user or application level there are four types of general purpose higher order level language/command capabilities needed for basic network operations. Here are some typical capabilities of distributed network services available from Type 1 vendors in Table I:

1. Remote file access and transfer—gives user full access to data files at any remote node in the network.
2. Downline loading—provides a host node the capability to downline load code to any satellite node for later use.
3. Remote command processing—allows user at any node to commence task execution on any other node in the network with similar set of operating system commands.
4. Program to program communication—the capability allows user application programs at one node to exchange data and control information with any program at another node. Each vendor has variations and supplementary provisions to use this capability in different ways such as equal partners which are dynamically variable or in master/slave arrangements.

There are many different ways to do these network functions between nodes but basically they consist of either file transfer or program-to-program operations. The particular way network operations are done depends on the application and/or user discretion. In the typical data reduction network scenario the user usually has the computer nearest the data storage (not necessarily the data acquisition computer) execute the program for data reduction. For example if a large data base needs to be searched on the host node to do some satellite node processing, then the program-to-program communication approach would logically minimize link traffic in having the host do the processing and later pass over the results to the satellite node. The reverse is also true; when large numbers of files (stored at the host node) are needed for local (or satellite) node processing, one would typically use remote file access and transfer files for use.

Remote command processing provides the much advertised advantage of distributed processing, namely, peripheral resource sharing. Each vendor implements his network software to provide the user with some degree of peripheral device transparency and/or device independent operations.

User transparency for tape and disk storage operations is fully supported but for other devices such as printers and displays, this may or may not be practical or desirable depending on the application.

Downline loading from a host node to a satellite node is the other obviously needed capability for any distributed network software. The two advantages for this are related to minimizing the number of development resources and minimizing equipment at the satellite node. For instance, only the code required resides in the satellite node but it can be changed and reloaded as often as necessary. There are several types or degrees of capability one might like in downline loading control and execution of jobs on remote nodes. Basically then, downline loading can move the system and user jobs to the satellite and start execution of same. An additional capability offered by DECNET, for instance, is to transmit an additional job to the satellite node while the previous jobs are running and have that new job start in its execution also on the satellite. So far no vendor supports overlay programs on the satellite from the host.

#### *Network software*

Distributed network node software is designed and implemented in a modular and layered fashion to facilitate added capabilities (i.e. growth and flexibility), and thus minimize impacts on modules or layers not directly changed. In a real sense it's a structured programming methodology. For obvious product considerations, the distributed network software functions are a separate yet compatible extension of the vendor's proven operating systems. Thus a user loads this distributed network software subsystem (usually designated by some clever acronym) in addition to or in conjunction with the operating system. From this point, each vendor develops his own implementation details for the distributed network software package.

Each Type 1 vendor in Table I offers special or unique features related to his equipment capabilities and those things found to be useful or necessary in many applications. An example of this is that Digital Equipment Corp. sells machines of 12, 16, 32 and 36 bit words and in order to network any combination of these they offer standard supported interface hardware and software to make these differences transparent to the user under DECNET. Hewlett Packard's Distributed Systems Network provides "store and forward" storage capability at each node. This facilitates routing and rerouting through the links if any link or node goes down. Modcomp has developed their networking system to improve performance by providing input-output operations concurrent with system operations using special (i.e. Modcomp calls "symbiont") software, and by implementing a "core device" option for direct CPU-CPU transfers which provides a much faster method than normal. Note that none of the first three vendors (DEC, HP, and Modcomp) built its network software for X.25 compatibility, but all three plan to develop compatible interfaces to their software for X.25 operations.

## MULTIPROCESSING ARCHITECTURES

Several minicomputer and midcomputer vendors provide several different kinds of capability for interconnecting multiple computers into multiprocessing systems. There are four Type 2 manufacturers offering bus interconnected hardware between CPUs. For Data General and Digital Equipment Corporation minicomputers, the user tailors the vendor's realtime operating system to his application using this hardware. The other two vendors supply turn key systems. Shared main memory products are offered by six vendors as presented in Table I. Table I, Type 4 presents the vendors offering shared disk hardware which just about includes all the vendors with this type of product. The user is responsible for operating system software changes to accommodate their application when using either shared memory or shared disks configurations.

#### *Multiple computer bus interconnected hardware*

Four manufacturers provide the computer hardware bus interconnected capability, and the features of these systems are discussed below. Using these hardware capabilities, as is, allows the user to achieve high performance utilization because the system can be tailored to their application. It also provides a means to tie together many computers in multiprocessor configurations with no special hardware development.

Data General was the first major minicomputer manufacturer to offer standard bus hardware for interconnecting up to fifteen machines for multiprocessing operations. Their standard interconnection hardware is designated as the Multiprocessor Communication Adapter (MCA) and was first introduced as a standard product back in 1970. It accommodates any of the Data General Nova or Eclipse line of minicomputers. The MCA in effect connects the direct memory access channels into a ring or daisy chain structure so data transfers can be made from any one CPU's memory to any other designated memory at very fast transfer rates.

Data General recently (1978) announced an additional type of bus interconnection capability called the Multiprocessor Communications System (MCS) which connects up to fifteen of their Novas and Eclipses into a star or radial bus network. It has the same data transfer rating of 1M bytes per second through direct memory channel interfaces. Buffering is provided internal to the MCS and special instructions control data transfers with variable sized block transfer capability. Also computers can be added or removed from the network without affecting overall operation.

#### **Data general**

One user implemented a 4 millisecond closed loop missile simulation using three Data General minicomputers in a MCA ring with their Model M600 modeling the missile, their Model S-230 simulating the target, and their Model S-130 being the signal generator control as shown in Figure 1. In

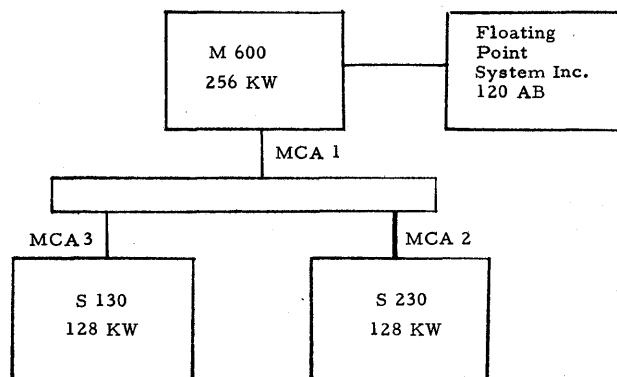


Figure 1—Computer simulator configuration using Data General's MCA bus

order to minimize software overhead on the bus and thus maximize throughput, this user coded his own MCA software drivers and attained an effective transfer rate of 100K to 150K words per second. This is the useful data rate range after overhead, contention, etc., on the MCA bus. All the software development and simulations are run under AOS 2.1, Data General's Operating System. As long as there is sufficient main memory capacity, the MCA bus seems more than adequate to maximize overall throughput.

### Digital equipment

Digital Equipment Corporation offers a Parallel Communication Link (PCL11-B) hardware option for ganging together up to 16 PDP 11 minicomputers with a common time shared bus. The PCL11-B hardware attaches to Digital's PDP 11 Series Unibus and if any PDP-11 does down, the others are not affected. User programs logically reference other CPUs on the PCL bus in a fashion similar to designating any peripheral device and PCL is supported under DECNET. Contention for the link is resolved by assigning a time slice to each PCL11 slot. A recent example utilizing this equipment is the Defense Communications Agency's Autodin II, a general purpose data communications network system. Autodin II operates in a packet-switching manner very similar to ARPANET. This multiple computer configuration (i.e., 11/70s) provides high availability by a special configuration which allows graceful degradation. The Autodin II network has eight nodes, each of which consists of 3 to 5 PDP 11/70s tied together with the PCL-11.

### Two new multiple CPU, bus interconnected architectures

A more recent trend in multiple CPU, bus interconnected hardware, is offered by two relatively new manufacturers (BTI Computer System and Tandem) which provide complete off-the-shelf user supported software with significant fail-soft capabilities. The manufacturer has to build into the operating system many features to accommodate the many different hardware malfunctions and resulting reassignment of resources. This makes it difficult for each user of such a

system to customize his operating system and still maintain the software integrity to support fail-soft operations. In other words, the user is encouraged to do everything in higher order level languages such as FORTRAN, COBOL, BASIC, PASCAL, others, and leave the manufacturer's operating system as is except for well-defined improvements that don't impact the fail-soft operation. These are new architecture implementations, each different, but offering capabilities every user should know.

### BTI 8000

The multiprocessing architecture approach taken by BTI Computer Systems Inc. in their new 32 bit machine, Model 8000, centers around their very high speed 32 bit bus (patented) which supports a 60M bytes per second bandwidth. Almost any combination of Central (i.e., computational) Processing Units, Memory Control Units, and Peripheral Processing Units (PPU) can be used in the fifteen available chassis slots (16th slot is required for the System Service Unit). For instance, if the user application is processor bound, then up to thirteen Central Processing Units could possibly be used. In input-output bound cases more Peripheral Processing Units could be installed. Some combination to match the user's loading can easily be accommodated using this highly flexible multiprocessing operation. The key point is that the 60M bytes per second bus should never be the bottleneck or the limiting part of this multiprocessor design. In a time sharing system, one PPU can support up to 256 user terminals each operating at speeds of up to 19.2K bits per second. The other significant aspect of interest in this architecture is that it supports true homogeneous multiprocessing operation where all resources (i.e., CPUs) are equal partners with no master-slave internal to the design. The designers of this system used PASCAL as the basis for much of the computer hardware design so the influence of PASCAL operations has driven much of the internal hardware design features as was originally done in the case of ALGOL for the Burroughs 5500.

The System Service Unit is strictly for the customer engineer to facilitate troubleshooting and repair of the entire BTI Model 8000. It has a small microcomputer (as do the MCU and PPU) which runs the diagnostics to determine faults and isolate them to the chip level. The SSU has its own telephone port for remote troubleshooting from the factory. After any major module goes down it is taken off-line; the user simply does a SYSGEN automatically by just pressing a button.

### Tandem T16 series

Tandem Computers Inc. installed its first multiprocessor system back in 1976, and last year the sales of this product were over \$25M with projections as high as \$40M this year. This noteworthy success of the Tandem T16 Series computers shows that up to sixteen tightly coupled minicomputers can replace a large scale computer with the equivalent



performance and offer additional advantages such as higher availability and more efficient accommodation of large numbers of disks and terminals. The Tandem Computer architecture consists of a high bandwidth (i.e., 26M bytes/sec) bus called Dynabus which provides interprocessor transfers separate from all the peripheral (i.e., disks, tape, terminals, etc.) transfers. These are done over their own direct memory access channel at speeds up to 4M bytes/sec. Interprocessor communication is provided with Dynabus; therefore, no shared main memory is required. The other key multiprocessing architecture success is their operating system (130K bytes per minicomputer processor) that provides complete user transparency to resource control and allocation, load balancing, fail-over, and many other functions.

For example, one Tandem Five processor system is used as a back-end file manager in Columbus, Ohio by the Ohio College Library Corp. It does library catalog processing in conjunction with forty 240 MB disk drives supporting several large host computers (i.e., Sigma 9s). Another Tandem dual processor front-end system is used to interface and operate the 2000 CRT alphanumeric terminals (each with a maximum 2 second response time limit) where each processor has 512K bytes of its own memory. Typically, if the system has a large number of page faults, the processors need more memory and 2M bytes is the maximum per processor. The multiprocessing environment is facilitated with stack oriented CPU design.

The Tandem non-stop or fault-tolerant capability consists of a wide variety of special hardware and software techniques. A basic system has two or more processors, two cable Dynabus peripherals with dual port controllers, and disk drives with dual access control as well as redundant disk drives. Tandem's specially designed disk controller has a 4K byte buffer to support recording to one or more disk drives without having to retransmit on the input-output bus. It also provides the capability for one disk unit to be copied to another without involving the input-output bus. Of course, there is a 10-30 percent throughput performance overhead to provide checkpointing and the necessary updating of file generation in pairs. Any operational module can be replaced on line without stopping operation of the Tandem System. There is also a dual power distribution in Tandem so if one power supply fails, the backup is automatically switched in.

### Shared main memory operations

Shared main memory provides capability for multiple CPUs to be connected to one shared memory. The advantage is that each CPU has access to the data in shared memory and typically users do not store executable code in shared memory. Shared memory addresses are established at SYS-GEN time typically as Global Common areas in the Fortran sense although it should be true for any language. The linking loader is used to connect logical names to physical addresses so user applications can use data in shared memory for those processing programs which are executed in local (i.e., not shared) memory. The beauty is that all the CPUs with a port on the shared memory module can access that data at very

fast main memory access speeds in a well regulated or disciplined method so as to minimize impact on overall processing throughput and also eliminate any need for multiple copies of data residing in each memory of every CPU needing the data for processing. With the cost of memory significantly decreasing all the time, the major benefit is definitely in achieving higher throughput with parallel processing. It does this by minimizing data transfers amongst multiple CPUs because all the CPUs have the necessary common data access.

### Perkin-Elmer

As an example, let's go through a typical application as to how a user really benefits from shared main memory capability. Assume the processing algorithms in each CPU take much longer to execute than the time for the buffer shared memory to fill from the source via the DMA channel. In order for the processing to keep up, multiple CPUs must have access to the data buffer (as shown in Figure 2). Without shared memory copies of the same data have to be stored in several memories so processing can be done in parallel with the resulting overall throughput being slower. Figure 2 shows data coming from some source in short bursts at very high speed data rates over a programmably switched direct memory access (DMA) channel to allow for multiple data sources. Figure 2 shows CPU A and CPU B sharing one memory module. For instance, with the Interdata 8/32 fourteen ported memory module, one could have fourteen (in any combination of CPUs and external devices) accessing one shared memory module. Some manufacturers can hook-up only CPUs to shared memory and not special external devices.

Perkin-Elmer uses the same hardware option (i.e., Local Memory Interface) to provide a parallel direct memory access path from disk to local memory for simultaneous transfer operations while data is transferred to the shared memory.

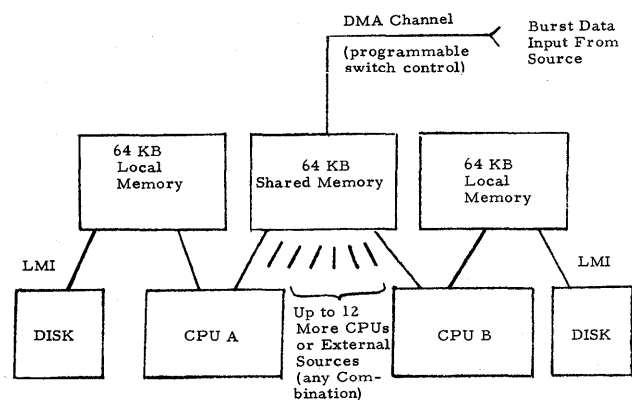


Figure 2—Perkin-Elmer's shared memory expansion capability

\*Abbreviations used: DMA—direct memory access; LMI—local memory interface

In addition, Perkin-Elmer has multiple memory banking where each memory bank operates independently of the other banks and multiple CPUs have simultaneous access to the different banks in the system thus increasing the overall throughput. A shared memory module can be very large (e.g. Perkin-Elmer 16M Bytes) where the memory porting is to/from the memory controller, not the memory module itself. Perkin-Elmer 16M Bytes shared memory is to/from four memory controllers capable of looking like one fourteen ported shared memory.

Any manufacturer offering multiport (i.e., shared) memory operations has to have a "test and set" instruction or the equivalent to facilitate accessing memory locations in a well disciplined manner and to preclude contention or conflicts with other CPUs accessing the same memory locations. In other words, the "test and set" instruction provides the necessary mechanism for software (i.e., program accessing memory) synchronization. For instance, the Perkin-Elmer's Test and Set (TS) instruction executes in only one instruction cycle and the built in hardware priority ensures that the other CPUs cannot interfere during the execution of the TS instruction. The TS instruction which executes within one instruction cycle first reads the first memory location in the buffer and writes a one in the significant bit position of the word to set a busy flag. If the busy flag has previously been set, then the condition code reflects this, and the CPU tries again later.

The other even more important aspect is how should contention amongst fourteen CPUs be handled in a well disciplined manner? There are several approaches to signal buffer ready or buffer available between CPUs. The simplest is by polling (i.e., testing if the flag is set or reset) the first buffer memory location with a test and set instruction periodically to determine if the buffer data is available for that CPU and then to proceed to read the buffer out. When only a few CPUs are on a shared memory, the controller allocates memory cycles in a round robin manner to access words. Another way supported by manufacturers is to use hardware interrupts. In this case, one CPU triggers an interrupt to another CPU when it has finished reading (i.e., relinquishing) the data buffer so the other CPU knows it can access the data buffer at that time. With a large number of CPUs such as fourteen, a priority scheme is used and the lowest priority job encounters some waits or delays. For instance Perkin-Elmer built into their memory controller both sequential access (i.e., round robin) and fixed priority with strapping to select one or the other. The third and most complicated way is to use some sort of data communication line which involves software message processing in effect from the CPU relinquishing the data buffer to the next CPU available to read the data buffer.

#### *Shared disk configurations*

Shared disk configurations are utilized to achieve significantly different benefits from those for which shared main memory configurations are implemented. One major appli-

cation of shared disk configurations is to provide large file storage for long term sequential processing steps where data is collected and updated on demand. An obvious example of this is online airline reservation systems. Data is really not reduced as in shared main memory applications but in fact is a data base which requires continuous up-dates for 24-hour-a-day operations. If one briefly considers what capabilities and support are necessary to safely have more than one CPU accessing data base files, it turns out you probably need to modify the existing operating system to achieve this. For example, in a standard operating system, a program may open a file, allocate 1000 blocks and begin writing records. Typical operating systems only write an EOF (end-of-file) when a file is closed. To conserve disk I/O during "append" operations, the number of blocks used, and indicated in the header, will be updated only at "file closed" time. So there is a problem that when the disk is suddenly switched to a new computer it sees 1000 blocks allocated and 0 blocks used. The EOF is in the first computer and not the on-disk. Thus a significant amount of overhead occurs in shared disk systems since EOFs must be written each time. In addition, many file systems block records to conserve disk I/O. Thus a program may write a half dozen records and the operating system will buffer it in main memory until a seventh is written and then write the block to disk. If the disk is switched or the computer accessing disk changes prior to the block being written to disk, the records blocked in main memory are lost. Also any records crossing block boundaries would be written to the shared files in an uncontrolled order so for these reasons block buffering cannot be allowed.

The benefit of shareable disk in the above case is to have reasonably fast response (i.e., high throughput) in accessing the data base, with power outages not causing data base to be lost. Even more popular, however, is higher availability because one computer going down does not bring the system down. Manufacturers now offer support with appropriate hardware (i.e., time-out logic) and operating systems to provide the necessary capability to accommodate switchover between CPUs in case one fails. This latter attribute is by far the most predominant capability now being sought by the user. Thus achievement of high reliability and graceful degradation involves some sort of shared or switchable disk capabilities. The distinction here is that shared disks are used in on-line transaction oriented systems whereas switchable disks are utilized in redundant or fail-over operations to facilitate graceful degradation of operations. Switchover can be done by manual operator control or under software control. These kinds of applications have typically used dual ported and even tri ported disk drives, which means separate disk controller paths exist to the disk drive electronics. In a fully redundant configuration, a failure in the CPU, channel, controller, or disk drive unit will not affect the proper operation of the other path. Note that dual port disk controllers (which accept data from either two different CPUs or from two channels of the same computer) can be a subset capability of shared disks but having dual ported disk controllers does not necessarily mean shared disk operations. The different types are shown in Figure 3.

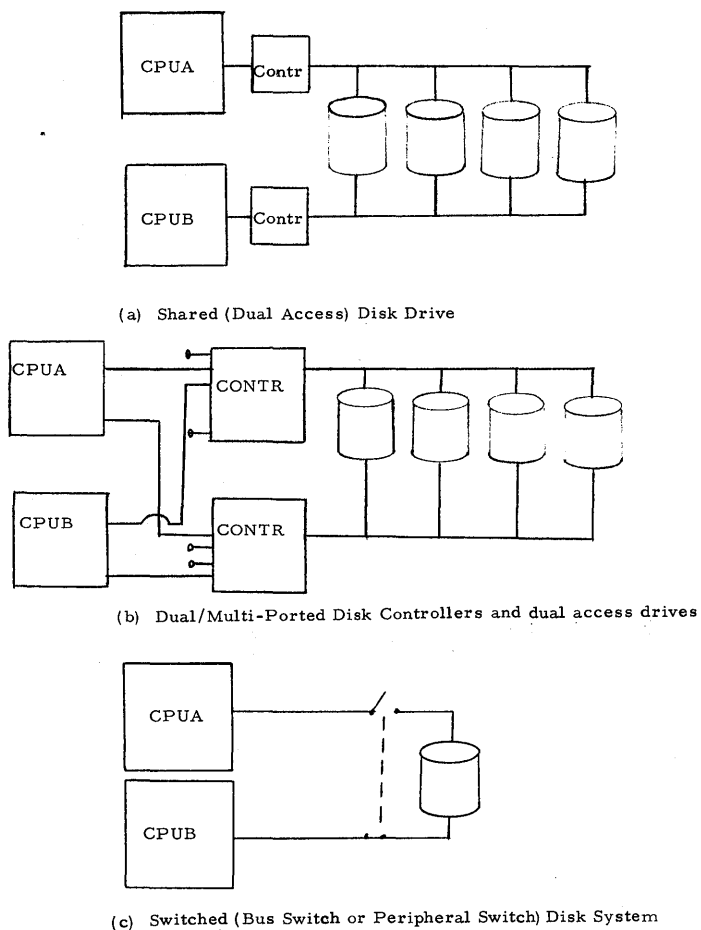


Figure 3—Shared/switched disk configurations

## RELATIVE COMPARISON OF NETWORK AND MULTIPROCESSING ARCHITECTURE TYPES

The network and multiprocessor implementation approaches are given in Table V. This is qualitative and not quantitative, although numbers or a range could be given for relevant user characteristics, such as average response time and average transfer rate. Each user has a different mix of criteria and weighting of each criterion, so it would be difficult to establish an overall rating system. What is useful, hopefully, is to identify the key performance advantage of each type and the inherent limitations or disadvantages of each type.

The distributed/network (Type 1) approach is available from several manufacturers where the user uses the off-the-shelf hardware/software support with no development required except to define the network configuration and resources and install the application software modules to do the user data processing functions. This approach gives the user off-the-shelf networking capability but at the expense of considerable general purpose capability that a user may

only partially need. In addition, special applications do not use all the software support to handle their kinds of network structures. However, this capability minimizes required user's knowledge of the details at the expense of the overhead to provide all these conveniences. Users can now rapidly implement (at lower cost) distributed/network applications, but at the price of slower data transfer rates and longer response times than with their own network architectures. Also, it is usually difficult to change or modify generalized distributed/network software when required.

Multiple CPUs interconnected with some form of high speed bus structure is a logical way to achieve fast data transfer rates for multiprocessing system. These vary as to design implementation specifics and hardware component technologies that are employed to maximize throughput. Multiprocessing in turn provides the user with faster response times for processing tasks or jobs. The disadvantage is that in order to achieve this high throughput performance, the manufacturer or user must develop specialized software to tailor his application and to maximally utilize these capabilities. Only that manufacturer supports the hardware and software, and it has no systems compatibility with other manufacturers. All the multiprocessing systems are restricted to one location but can support terminals remotely over communication lines.

The major advantage of shared main memory via multiports is the almost instant accessibility by many CPUs to data where the processing time is much longer than the data collection time. Shared memory can thus be used to achieve extremely fast throughputs. It drastically reduces overhead in both timing and storage requirements by providing common access to many CPUs concurrently at semiconductor cycle times and by not having to duplicate copies of data in many memory banks. There is also a hardware cost savings in that shared memory architecture requires no channel hardware which is typically in short supply to support all the peripheral devices.

The last type of approach to linking together computers is a loosely coupled arrangement via the shared disk. This allows each computer to process jobs with no special timing or synchronization between machines. In other words, shared disk operations are asynchronous. For this reason the operating system has to be modified and the file manager system is impacted depending on the application and the file security and protection required. Shared disks typically use high speed direct memory access channels for reading and writing files, but, of course, this is slow compared to shared main memory access times. The disk, however, provides much larger storage capacities for two orders of magnitude lower cost per byte. Also disk data is nonvolatile when the power goes off which is a disadvantage of semiconductor shared main memory without batteries. Shared main memory supports as many CPUs as there are ports on the memory module, whereas dual and triported disks are about the practical limit as a shared resource. And last, but not least, just about every manufacturer in the business offers shared disk hardware capability.

TABLE V—Comparison of four computer configuration types

	Type 1 Distributed/Network	Type 2 Multiple CPUs Interconnected With Bus Hardware Configurations	Type 3 Shared Main Memory	Type 4 Shared Disk
A D V A N T A G E S	<p>Several manufacturers offer complete off-the-shelf operating system support</p> <p>Distributed/network system easy to implement with no systems development just application software</p> <p>Some manufacturers support any network configuration (e.g., star, ring, n-level hierarchy) with both hardware and data communication links</p>	<p>Very fast transfer rates</p> <p>Number of CPUs limited only by adapter hardware design</p> <p>Processing response time is inherently faster than other types of configurations</p>	<p>Very fast and convenient access to data for many CPUs</p> <p>Number of CPUs limited only by the number of memory ports</p> <p>Saves by eliminating storage of common data</p> <p>Minimize input-output operations (i.e. no channels required)</p>	<p>Minimizes impact on computer operations especially timing</p> <p>Timing/synchronization requirements relatively easy to establish</p> <p>Shared disk uses direct memory access channels for high transfer rates</p> <p>Requires standard data base file/record formats between CPUs</p> <p>Most manufacturers offer this hardware capability</p> <p>Large capacity, non-volatile</p>
D I S A D V A N T A G E S	<p>Higher operating system overhead for generalized design</p> <p>Slower transfer rates and longer response times typically</p> <p>Difficult to change generalized design hardware and software</p>	<p>Usually not compatible with data communication link protocols</p> <p>Very specialized operating system software support</p> <p>Application flexibility is limited by software and hardware design system peculiar to the manufacturer offering this capability</p> <p>Uses hardware cables to interconnect and there are cable distance limitations</p>	<p>Special instructions and software support has to be used</p> <p>All installations have differences so manufacturer has only partial software support package</p> <p>Common data capacity limited to capacity of the main memory address structure</p> <p>Only a few manufacturers offer this capability as yet</p>	<p>Response time to access file is much slower 100 milliseconds versus 10 microseconds for shared main memory</p> <p>Disk storage overhead is high</p> <p>Limited to three CPU accessibility per shared disk</p> <p>Special software development expense</p>

## FUTURE TRENDS

We have every expectation that distributed network and multiprocessing types of architectural approaches will continue to increase in terms of the manufacturers offering this support and their use in application implementations. The reasons vary for why each type will evolve with more and more uses, and the trend seems clearly established already (we have only begun to see the tip of the iceberg). Distributed/network systems are clearly evident in all kinds of business operations with remote nodes feeding a hierarchy or a centralized node to process data orders, update inventory and provide all the many functions described in so many other articles. Their future looks boundless and unlimited.

As more users begin to realize that multiprocessing capabilities exist and that one has only to take a sound, prac-

tical approach to implementing an architecture to map those resources to their application, the risk of multiprocessing will remain a memory to only the old timers in the business.

The high speed bus type of multiprocessing architecture which provides automatic fail-soft or fail-over capability will emerge into its own type and have its own marketplace for products to compete with large scale, single computer machines. Many users will want this because it offers significant cost advantages with low acquisition cost and no staff of systems programmers needed to support operation. The last type is shared disk system which will be used less for purely multiprocessing purposes because shared main memory costs are decreasing all the time and more for large data base applications. But use of shared or switchable disks will steadily increase to accommodate fail-soft or fail-over capability to provide users the high availability and reliability business demands.



# ARQ performance in SNA networks

by MARTIN A. REED

IBM Corporation  
Gaithersburg, Maryland

and

TERENCE D. SMETANKA

IBM Corporation  
Research Triangle Park, North Carolina

## THE SNA NETWORK

In order to examine various data link control (DLC) procedures, one must first have a basic understanding of the overall network. The network depicted will be one implementing the IBM Systems Network Architecture (SNA).<sup>1,2,3,4,5</sup> The fundamental communication system within SNA is referred to as the Transmission Subsystem. The Transmission Subsystem includes three types of elements. The first element, Data Link Control (DLC), transfers packets (or SNA Basic Transmission Units) across noisy transmission mediums. The protocol used here is that of SDLC (Synchronous Data Link Control). The second element, referred to as Path Control (PC), routes packets either to an end user in the node or to the proper DLC element for transmission back through the network. Path Control also blocks incoming messages and deblocks outgoing messages (i.e., SNA segmenting). Transmission Control (TC) helps manage SNA session (i.e., connection) initiation and termination, provides sequence number manipulation, controls pacing, and performs many other functions on behalf of the end user. Pacing is a means of controlling the rate at which TC sends and receives normal data flow requests.

The SNA Function Management Services provide for the control of data flow and for transformation of data presented to the network. One element of FM Services is that of Data Flow Control (DFC). DFC is used to handle chains of related requests, the modes of data requests and responses, and other flow control procedures. A chain represents an end user's breakdown of a basic work unit. Another FM Services element is Presentation Services (PS) which provides support for communications between end users engaged in sessions. PS is a type of mapping service which can adapt various end users' (terminal operator or application) interfaces to the SNA network.

The various SNA layers described allow the attachment of Network Addressable Units (NAU's) to the network. Three NAU's are defined. The Logical Unit (LU) provides a means for end users to interface to the network. The Physical Unit (PU) is a component of each communications re-

source (e.g., host, communications controller, cluster controller, or terminal). The third NAU, System Services Control Point (SSCP), is the brain of the network. The SSCP is responsible for the general management of the network. The Physical Unit containing the System Services Control Point (e.g., 370, 3033, etc.) is referred to as a PU Type 5 (PU.T5). The communication controllers (e.g., 3705) are referred to as being PU.T4. Cluster controllers (e.g., 3274, 3276, etc.) are referred to as PU.T2's and terminals as PU.T1 (e.g., 3767).<sup>1</sup>

With the various SNA layers and NAU's described,<sup>2,3</sup> it is now possible to describe session initiation and termination. A session between LU's is simply the connection whereby packets flow between the LU's as part of a related series of transactions. Figure 1 depicts session initiation among two LU's in the same domain (i.e., controlled by the same SSCP). Either the Primary LU (PLU) or the Secondary LU (SLU) sends an INITIATE command to the SSCP with the network name of the requested LU. The SSCP resolves the LU name and initiating origin into network addresses and places the BIND image as well as network addresses into the CINIT which is sent to the PLU. The BIND image contains information pertaining to various protocols which are to be used for the duration of the session. Such information includes chaining protocols as well as modes of requests and responses. The PLU then sends the BIND to the SLU and awaits SLU acknowledgment. On positive acknowledgment, the PLU sends SESSST to the SSCP in order to notify the SSCP of session activation. Session termination for LU's in the same domain is similar in flow to that of session initiation (see Figure 2 for SLU termination). The SLU sends TERMINATE to the SSCP. This command contains the name of the PLU with which the terminating SLU wishes to terminate this session. The SSCP then sends CTERM to the PLU requesting that the PLU attempt to deactivate the session with the specified SLU. This leads the PLU, upon acceptance of the CTERM, to send UNBIND to the SLU. Upon receipt of the UNBIND positive response, the PLU notifies the SSCP of session deactivation via SESS/END. For PLU termination, the PLU need only send

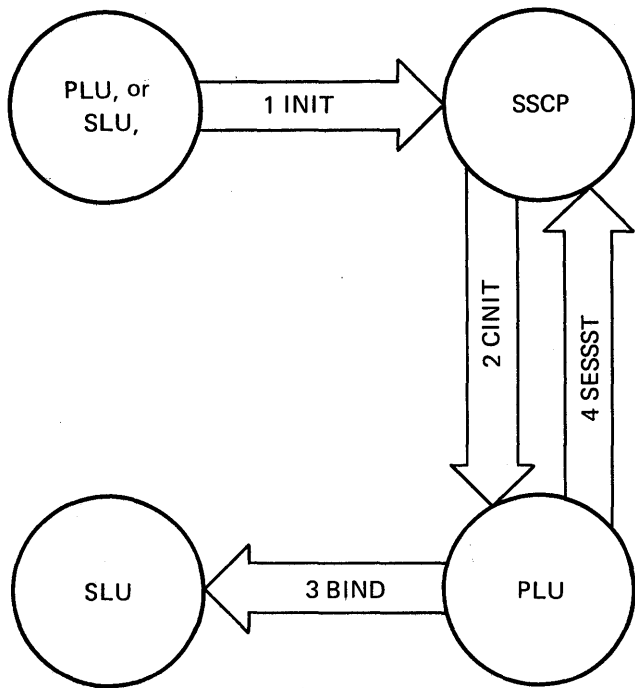


Figure 1—Session initiation.

UNBIND to the SLU. Upon receipt of the UNBIND positive response, the PLU sends SESSEND to the SSCP. It is also possible to have multiple SSCP's in the network, each managing its own domain. The initiation and termination of sessions among LU's in different domains is slightly more complex and is a superset of the single domain case.<sup>4,5</sup>

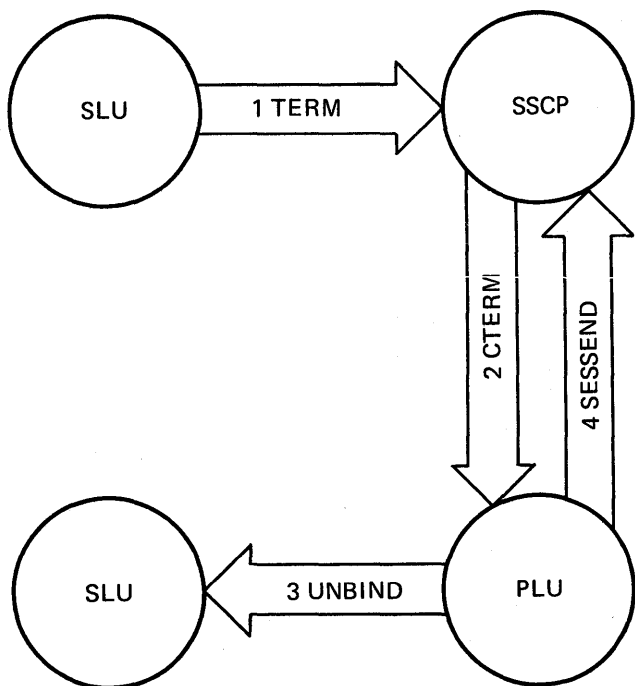


Figure 2—Session termination.

With the SNA session initiated, it is now possible to trace a transaction through the network as seen in Figure 3. One configuration is where two PU.T4's are between the two end users (e.g., terminal operator and host application). Here, the terminal operator's message traverses PS, DFC, TC, PC, DLC, and is thus transmitted over a common carrier link (or possibly an in-plant line if the communication controller and terminal are local to each other) to the communication controller. The message is routed through this PU.T4's DLC to its PC where it is determined that the message need be routed to the next node which is another PU.T4. Thus, the message is again passed through DLC and the common carrier link to the second PU.T4. The message is again sent to PC from this PU.T4's DLC where the routing tables determine that the message is to be routed to the locally attached host. Therefore, the second communication controller routes the message through its channel DLC to the host (PU.T5) channel DLC. The host PC determines that the message is for a host application and thus routes the message through TC, DFC, PS, and finally to the end user application. It is also possible to have only one PU.T4 in the path between the terminal and host as depicted in Figure 3. In fact, any number of PU.T4's may be between the terminal operator and the host. The common carrier as described above can be terrestrial links (e.g., telephone cables, undersea cables, microwave links, etc.) as well as satellite links. The DLC pairs between any two nodes manage the retransmission

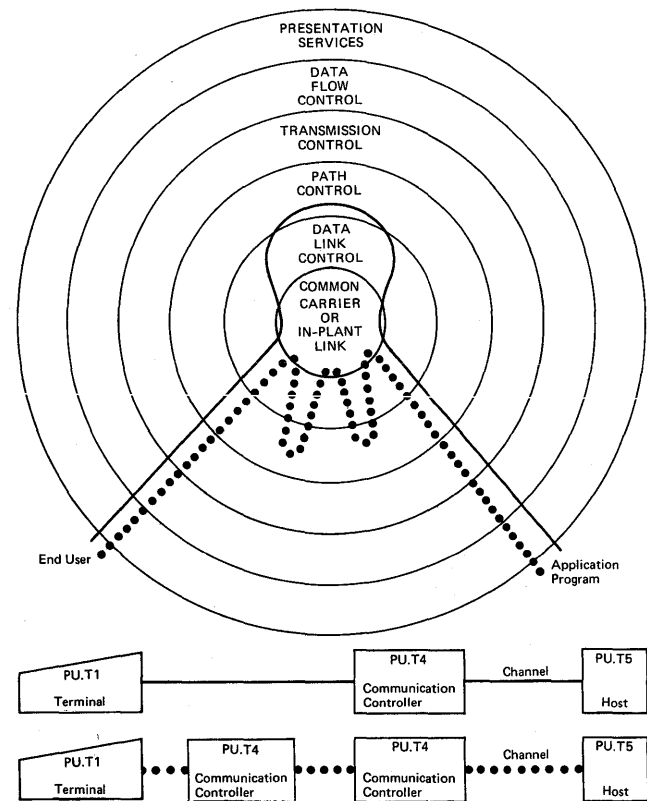


Figure 3—Data flow through network.

schemes necessary to assure that the message is sent correctly over the noisy data links.<sup>2,3</sup>

### THE DATA LINK CONTROL FRAME

Performance of the network is critical to the interactive end user. As seen in Figure 3, tuning the network to achieve optimal performance should be done with consideration of each of the SNA layers. For example, the choice of definite response or exception response in Data Flow Control may be crucial to the overall response time or system utilization. Also, the length of chain elements is correlated to performance results. The previously mentioned pacing parameters in Transmission Control as well as the segmenting parameter in Path Control may become crucial to performance as well. Possibly the most difficult layer of SNA to tune for performance is that of Data Link Control. Data Link Control performance is not only dependent on such factors as transmission speed, propagation delays, and frame size, but also on the random variable of bit-error-rate (BER). The error rate of the link is a function of weather conditions, noise characteristics of power amplifiers, and other random factors, as well as deterministic influences such as line conditioning, which lead to the receiver obtaining a garbled message. Knowing how DLC fits into the overall network, it is now possible to examine this element in more detail to help the user achieve better response time.

SNA incorporates the use of Synchronous Data Link Control (SDLC) for line control.<sup>6,7</sup> SDLC is a subset of the International Standards Organization's High-Level Data Link Control (HDLC)<sup>8,9</sup> and the American National Standards Institute's Advanced Data Communications Control Procedure (ADCCP).<sup>10</sup> Any message may consist of one or more SDLC frames which each contain up to six fields (see Figure 4). All the fields are fixed in size excluding the information field which is a variable number of eight bit bytes. The two flags (01111110) enclose the SDLC frame. In order to assure data transparency, SDLC procedures require that a binary zero be inserted by the transmitter after any succession of five contiguous ones in the frame. The receiver removes these inserted zero's upon frame receipt. This zero insertion assures that only the two flags may have six contiguous one bits. The analysis conducted pertains to a data link running in unbalanced (i.e., containing both fixed primary and secondary stations) and normal response mode (i.e., polling is used rather than contention). The address field distinguishes the secondary station from which or to which the message is being sent. The control field contains polling information, sequence numbers, and commands and responses required to control the data link. Two subfields of the control field contain the sending  $N(s)$  and receiving  $N(r)$  sequence numbers which are necessary in double-numbering. In double-numbering, the sending station transmitting information frames inserts the sequence number of each frame in the  $N(s)$  subfield. In conjunction, the receiving station maintains the receiving count in the  $N(r)$  subfield.  $N(r)$  is incremented upon the receipt of each error-free frame, as long as the received  $N(s)$  matches the  $N(r)$  count. The  $N(r)$  count is

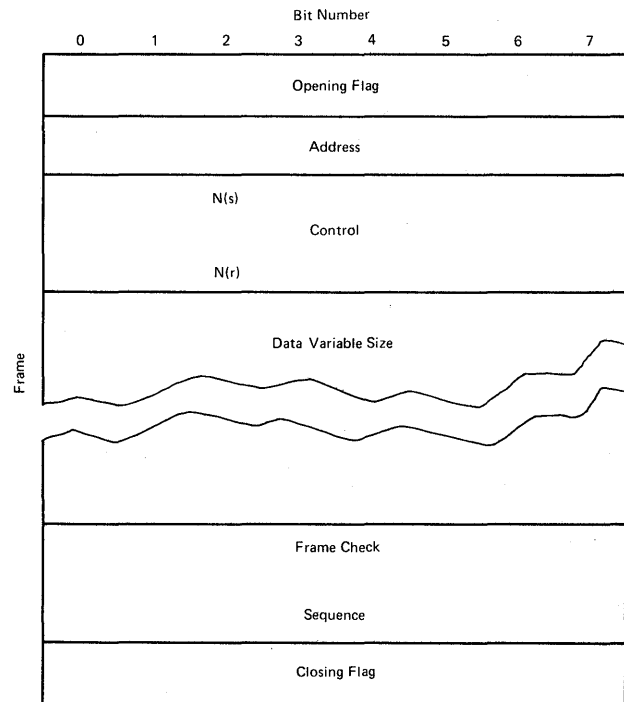


Figure 4—SDLC frame structure.

therefore the number of the next expected frame to be received. The  $N(s)$  and  $N(r)$  subfields are used by various retransmission schemes. The information field contains the user data. The 16 bit frame check sequence field is used in a cyclic redundancy check to determine if the frame was received correctly.

### ARQ TECHNIQUES

The choice of ARQ (automatic request for repeat) techniques may be an important consideration in regards to system performance. With the onset of satellite data communications, the choice will become even more crucial. Four DLC retransmission techniques will be analyzed. REJ (reject) and SREJ (selective reject) are architected ARQ techniques in SDLC<sup>7</sup> while SACK (selective acknowledgment) is an alternative technique which is not architected in SDLC. The fourth technique to be analyzed is that of BSC (Binary Synchronous Communications) which is a non-SNA DLC approach. Only one "frame" is allowed to be outstanding in BSC.

The first technique, REJ, can lead to the retransmission of non-error frames as well as those frames in error (as detected by the frame check sequence). The supervisory format of an SDLC frame is used to transmit an error control frame from the station detecting a sequence error in the received  $N(s)$  count. Retransmission is required beginning with the information frame where the  $N(s)$  count matches that of the  $N(r)$  count received in this supervisory frame. Frames  $N(r) - 1$  are acknowledged. All frames pending transmission



following the frame in error may be transmitted. Only one reject exception condition may be outstanding at any given time between the two SDLC stations. The modulo count-1 is the number of unacknowledged frames allowed to be outstanding. In SDLC, the modulo count can be a maximum of 8 while in HDLC a modulo count as great as 128 is possible. Therefore, seven frames may be sent under SDLC before acknowledgment is required. Figure 5 exemplifies REJ.

The second alternative, SREJ, only requests retransmission of the SDLC frame in error. Again, the supervisory format of an SDLC frame is used in order to notify the sending station of a sequence error in the received N(s) count at the receiving station. The single information frame to be re-sent is specified with the N(s) count specified in the N(r) count of the SREJ acknowledgment. The sending station can then retransmit the frame in error as well as any frames which have not been previously sent. The number of frames outstanding must be less than the SDLC modulo count. If more than one information frame is received in error, only one error frame can be re-sent on the following transmission (see Figure 5).

The third scheme is not architected in SDLC or HDLC at the present time. SACK is similar to SREJ in that only frames in error need be retransmitted. Unlike SREJ, all information frames received in error can be retransmitted on the following retransmission. Also, the receiving station must realize how many frames were received in error on the previous transmission in order to properly resequence the frames. Here, each frame re-sent is issued a new N(s) count in order to allow more frames to be sent on each transmission. SACK then has the capability of transmitting the entire sequence of frames in fewer transmissions (see Figure 5).

PERFORMANCE RESULTS

It is assumed that the network component under study consists of a half-duplex (HDX), point-to-point transmission link, operating either in BSC or in HDLC unbalanced normal response mode. This general description was chosen to allow consideration of both SNA and non-SNA networks. The analysis is equally valid for an SDLC environment; however, HDLC was chosen to allow the modulo count to range up to 128, rather than restricting it to 8 as is implemented under SDLC.

Interactive applications, rather than batch, are assumed to be the primary job types utilizing the transmission facility. Consequently, link response time is the key performance parameter. Previous studies<sup>11,12,13,14,15</sup> have described the analyses of the REJ and SACK modes of error recovery. BSC is conveniently handled by considering it as a subset of ARQ. SREJ has not been previously analyzed; its results appear herein for the first time.

The mathematical model of mean data link response time (MLRT) for BSC, REJ, SREJ, and SACK is summarized in the Appendix. Note that the input variable *s* (data frame size) includes both information bits as well as framing characters. The other input parameters are self-explanatory, except for the factor *t*, which accounts for network delays not associated with the actual transmission of user data. Consider a satellite link with terrestrial tails as a sample environment. A sequence of frames (from one to the modulo count—1) being sent from the secondary to primary station first incurs a delay to physically pass through the modem at the source of the data (modem transit time). Next, there is a propagation delay along the terrestrial tail from the secondary to its associated earth station, and a propagation delay along the satellite uplink. Any necessary satellite processing then takes place, followed by another set of delays on the completion of the trip to the primary station. Here, an acknowledgment frame (containing no information field) is constructed and transmitted back to the secondary station, incurring each of the described overhead delays on the way. The next transmission of data frames from the secondary station can then be initiated. Thus, the response time for one round-trip consists of the actual transmission time for the data frames plus the overhead time, which, in this case, equals four modem transit delays plus four terrestrial propagation delays plus four satellite propagation delays plus the transmission time for the acknowledgment plus any satellite processing time, as well as any configuration dependent delays such as waiting for a poll. The link response time for an entire message is the sum of these round-trip times for all transmissions.

Figure 6 illustrates the two components of MLRT for BSC, REJ, SREJ, and SACK. The satellite/terrestrial scenario has been maintained here, as can be seen in the large overhead delay, in order to highlight the differences among the techniques. Notice that BSC is independent of the modulo count, as is clear from its definition. SREJ and SACK both have constant transmission time components, since only errored frames are retransmitted, but have monotonically decreasing

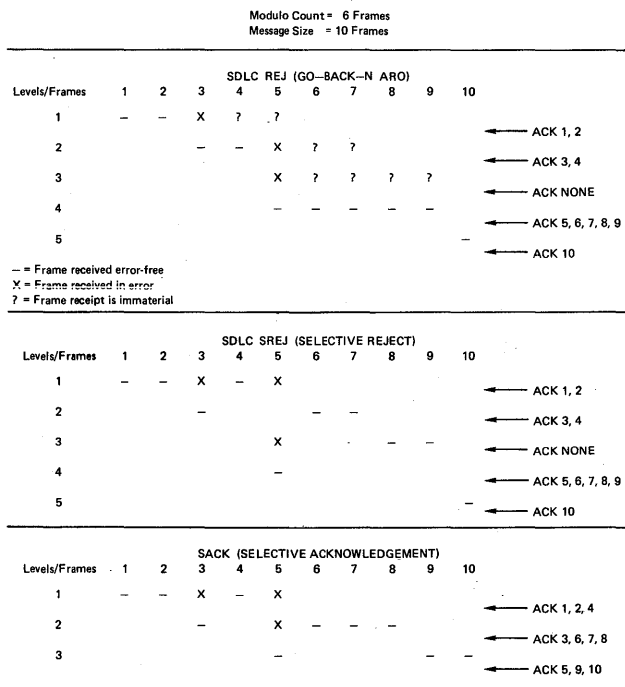


Figure 5—Examples of REJ, SREJ, and SACK.

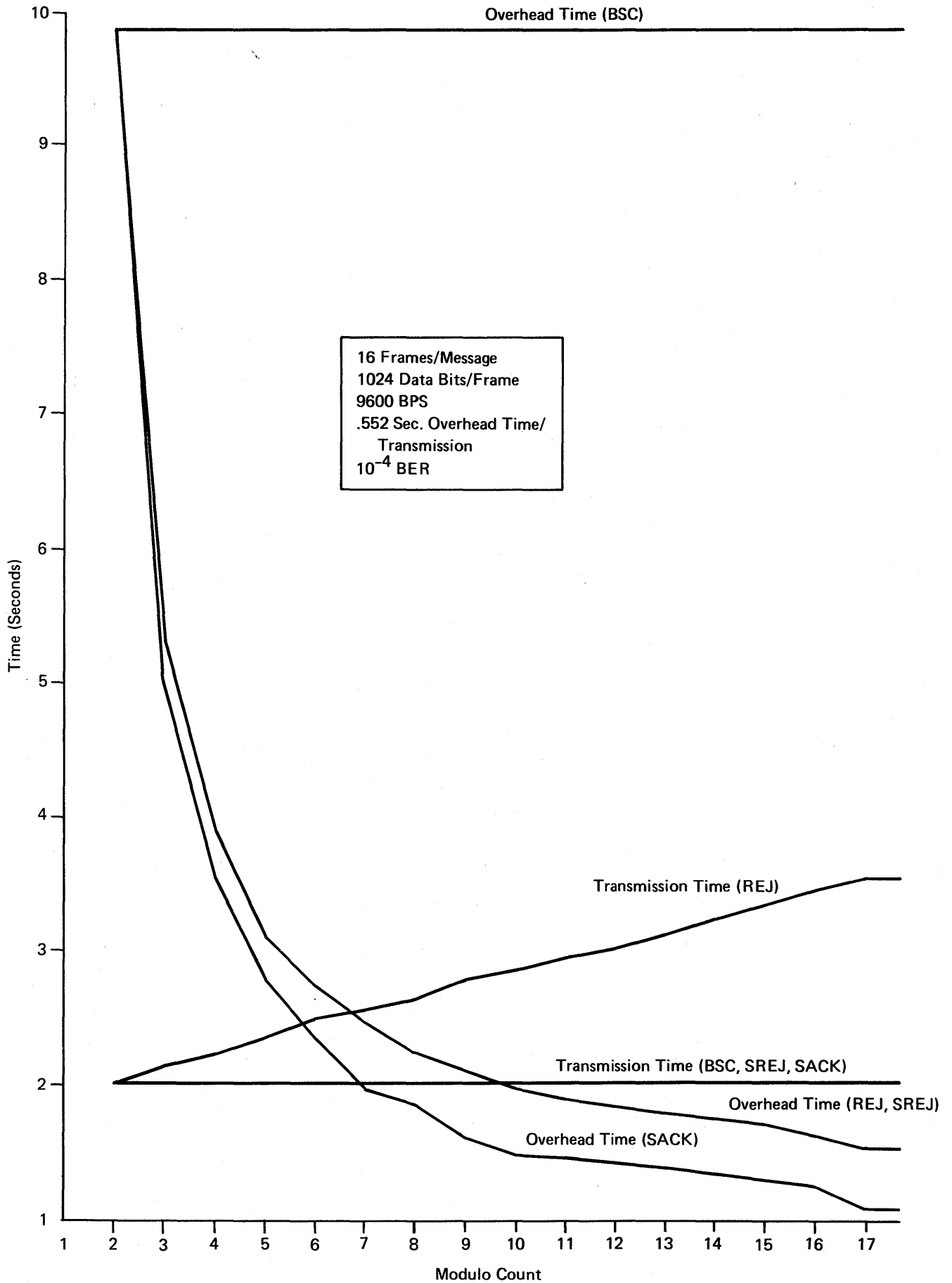


Figure 6—Components of MLRT vs. modulo count.

overhead time components, since fewer transmissions are required as the number of frames sent per transmission is permitted to increase. REJ exhibits this same behavior in regards to overhead time. However, the transmission time component of REJ increases with the modulo count, since more non-errored frames must be retransmitted for each errored frame received.

Figure 7 shows the resulting MLRT for the same benchmark. The error-free case is presented for comparison purposes. Observe that an optimization of MLRT for REJ as a function of modulo count can be realized. In contrast, the response times for both SREJ and SACK exhibit monotonically decreasing behavior as modulo count increases. A graph such as this can be used not only for performance specifications, but for systems design. For example, since buffering requirements at both the primary and secondary stations are a direct function of the modulo count, a tradeoff of response time vs. necessary buffering capacity can be made.

There are many design parameters which require study in the planning stages of a system. Hardware features such as data transmission rate of the link and bit-error-rate can easily be tuned from a performance standpoint by use of the model described here. In addition, software decisions can be reached. As an illustration, Figure 8 describes the effects on MLRT when the system from Figure 7 is varied by blocking the same 16,384 bit message into 8 frames rather than 16

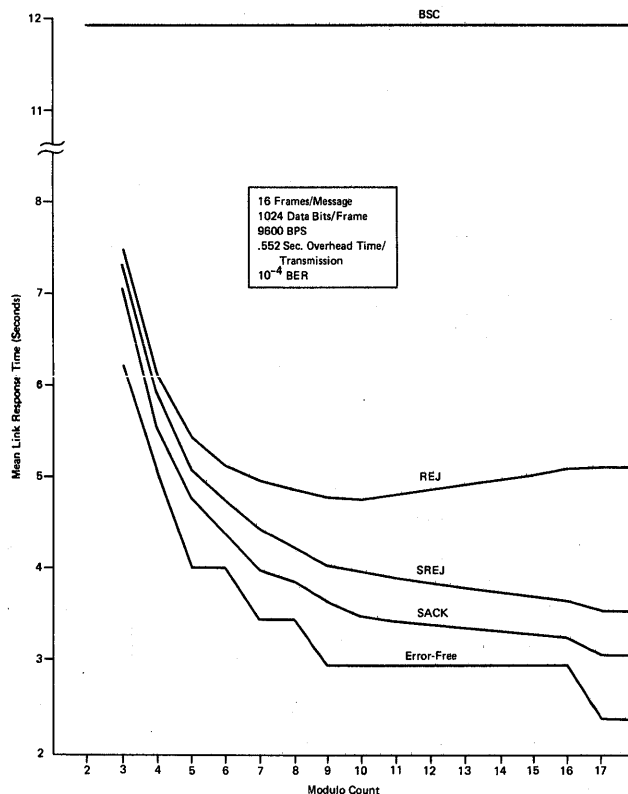


Figure 7—MLRT vs. modulo count at 16 frames/message.

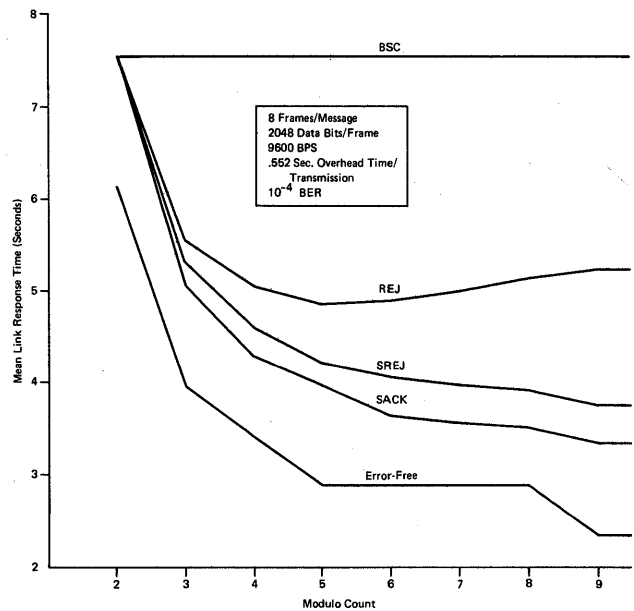


Figure 8—MLRT vs. modulo count at 8 frames/message.

frames. A different modulo count, buffering requirement, or retransmission technique may now become optimal.

CONCLUSIONS

The analysis of the DLC layer of a network is an intricate process. This paper has described an important subset of that effort, namely the study of data link response time for interactive applications using a noisy telecommunications link. All the presently architected ARQ techniques, as well as one proposed retransmission scheme, have been considered in the formulation of a mathematical model of the network. By use of this model, one may investigate the influences of many hardware and software parameters, thereby aiding in the planning, design, and cost/performance analysis of a system.

REFERENCES

1. Ahuja, V., "Routing and Flow Control in Systems Network Architecture," *IBM Systems Journal*, Vol. 18, No. 2, pp. 298-314 (1979).
2. Green, P. E., "An Introduction to Network Architectures and Protocols," *IBM Systems Journal*, Vol. 18, No. 2, pp. 202-222 (1979).
3. "Systems Network Architecture," General Information, IBM Publication GA27-3102-0, Mechanicsburg, Pa. (1975).
4. "Systems Network Architecture Reference Summary," IBM Publication GA27-3136-2, Mechanicsburg, Pa. (1978).
5. Gray, J. P. and McNeill, T. B., "SNA Multi-System Networking," *IBM Systems Journal*, Vol. 18, No. 2, pp. 279-282 (1979).
6. Kearsy, J. R., "Synchronous Data Link Control," *Data Communications*, pp. 49-60, May/June (1974).
7. "IBM Synchronous Data Link Control," General Information, IBM Publication GA27-3093-0, Mechanicsburg, Pa. (1974).
8. "Data Communications—High Level Data Link Control Procedures—Frame Structure," International Organization for Standardization (ISO)

- Document No. ISO 3309, American National Standards Institute, New York (1976).
9. "Data Communications—High Level Data Link Control Procedures (independent numbering)," International Organization for Standardization (ISO) Document No. ISO/DIS 4335, American National Standards Institute, New York (1976).
  10. Carlson, D. E., "ADCCP—A Computer-Oriented Data Link Control," *COMPCON*, pp. 110-113, Sept. 9-11 (1975).
  11. Reed, M. A. and Smetanka, T. D., "How to Determine Message Response Time for Satellites," *Data Communications*, pp. 42-47, June (1977).
  12. Smetanka, T. D. and Reed, M. A., "Error Considerations in Determining Satellite Data Link Response Time," *NTC 77*, IEEE No. 77CH1292-2 CSCB, 3B.3.1-3B.3.5 (1977).
  13. Smetanka, T. D., "Coping with Data Transmission Errors," *Mini-Micro Systems*, pp. 74-75, July (1978).
  14. Reed, M. A. and Smetanka, T. D., "Implications of a Selective Acknowledgment Scheme on Satellite Performance," *IBM J. Res. Devel.*, Vol. 23, No. 2, pp. 189-196, March (1979).
  15. Reed, M. A. and Smetanka, T. D., "A Mathematical Model of a Data Link Error Control Scheme for Response Time Distributions," *ICC 79*, IEEE No. 79CH1435-7 CSCB, 41.3.1-41.3.5 (1979).

## APPENDIX

### The mathematical model for data link response time

The mean data link response time MLRT is computed as follows:

Define the parameters:

$$p \triangleq 1 - (1 - \text{BER})^s$$

$$q \triangleq 1 - p$$

$$C(x, y) \triangleq \binom{x}{y}$$

$\lceil x \rceil \triangleq$  Least integer greater than or equal to  $x$

$\lfloor x \rfloor \triangleq$  Greatest integer less than or equal to  $x$

where the network and configuration parameters used to generate MLRT are:

$s$  = Data frame size, in bits

BPS = Data transmission rate of the link, in bits per second

BER = Overall bit error rate of the link (randomly distributed)

$n$  = Number of data frames per message

$M$  = Modulo count - 1

$t$  = Overhead time incurred in each round-trip transmission of frames from the primary station to the secondary, and back to the primary. This parameter includes such factors as modem transit delays, terrestrial and/or satellite propagation delays, delay for receipt of each acknowledgment, queuing delays, etc.

Then for each retransmission protocol:

#### BSC (binary synchronous mode)

$$\text{MLRT} = ((s/\text{BPS}) + t)(n/q)$$

#### REJ (reject mode)

$$\text{MLRT} = \begin{cases} n(s/\text{BPS})(1 + (p/2q)(n+1)) + t(1 + np/q) & \text{for } M \geq n \\ (s/\text{BPS}) \left( \sum_{l=\phi}^{\lfloor (n-1)/M \rfloor} q^{lM} \cdot \text{MIN}(M, n-lM) \right) + t((1 - q^{M(1 + \lfloor (n-1)/M \rfloor)}) / (1 - q^M)) \\ + \sum_{l=1}^{\infty} \sum_{x=1}^{\text{MIN}(n, lM)} \sum_{j=\text{MIN}(1, \text{MAX}(0, x - (M-1) - 1))}^{x/M - 1} C(l, j) p^{l-j} q^{x-1} \sum_{k=0}^{\lceil x/M \rceil - (j+1)} (-1)^k \\ \times C(l-j, k) C(l-j+x-M(j+k)-2, l-j-1) \\ \cdot ((s/\text{BPS}) \text{MIN}(M, n-x+1) + t) & \text{for } M < n \end{cases}$$

**SREJ (selective reject mode)**

$$\text{MLRT} = \begin{cases} (s/\text{BPS})(n/q) + t(1+np/q) & \text{for } M \geq n \\ (s/\text{BPS})(n/q) + t((1 - q^{M(1 + \lfloor (n-1)/M \rfloor)}) / (1 - q^M)) \\ + t \sum_{l=1}^{\infty} \sum_{x=1}^{\text{MIN}(n, lM)} \sum_{j=\text{MIN}(1, \text{MAX}(0, x - l(M-1) - 1))}^{\lfloor x/M \rfloor - 1} C(l, j) p^{l-j} q^{x-1} \sum_{k=0}^{\lfloor x/M \rfloor - (j+1)} (-1)^k \\ \times C(l-j, k) C(l-j+x-M(j+k)-2, l-j-1) & \text{for } M < n \end{cases}$$

**SACK (selective acknowledgment mode)**

$$\text{MLRT} = \begin{cases} (s/\text{BPS})(n/q) + t \sum_{i=1}^n (-1)^{i+1} C(n, i) (1-p^i)^{-1} & \text{for } M \geq n \\ (s/\text{BPS})(n/q) + t \sum_{l=\lfloor n/M \rfloor}^{\infty} \sum_{z=\lfloor n/M \rfloor - 1}^{l-1} \sum_{x=\text{MAX}(1, n-zM)}^M \sum_{y=M+1}^{x+M} l p^{zM - (n-x)} q^{n-x} \\ \times C((z-1)M, n-y) C(M, y-x) \cdot ((1-p^{l-z})^x - (1-p^{l-z-1})^x) & \text{for } M < n \end{cases}$$

# Computer communication in NTT remote computing services

by MASATOSHI IWAYAMA and ATSUMU FUJIWARA

*Nippon Tel & Tel Public Corp.*  
Tokyo, Japan

## INTRODUCTION

Today, NTT's RCS has higher-level and more varied functions than when it started. It occupies a more and more important position in computerization in Japan.

In this situation, in order to construct a rational service system according to user needs and to achieve an effective functions distribution between NTT's RCS and other Hosts (computers used by other service vendors or private company computers), computer communication techniques have become indispensable in NTT's services.

This paper first indicates requirements for computer communications in NTT's services. Second, it indicates the basic computer communications functions, needed to respond to those requirements and how far they have been realized by now. Furthermore, an overview of the protocols realized so far is reported and some of their technical aspects are discussed.

Last of all, the NTT schedule from now on is presented.

## NTT REMOTE COMPUTING SERVICES FEATURE

Since the utilization of computers began in Japan, the NTT concept has been that various computer systems and networks should be organically combined in the future to become a communication and data-processing utility (called a Network Utility) as a kind of infrastructure.

NTT believed that, in order to construct a rational and sound Network Utility in Japan, it was desirable for NTT, the common carrier, to enter upon the data-processing field. On the basis of this concept, NTT has been developing and offering a number of data-processing services since 1968.

These services are classified roughly into the two categories. One is the sole use data-processing service for specific users. The other is remote computing service (RCS) for unspecified users. RCS is divided into the two service categories, DRESS, and DEMOS. Both services have expanded gradually since they started in 1971. They have become the largest RCS in Japan with their 6,000 terminals.

DEMOS (Demenkoshu Multiaccess Online System) is a general purpose TSS (Time Sharing System) service with a multitude of commands, languages, library programs and

large processing capacity. DRESS (Dendenkoshu REaltime Sales management System) is a service for a fixed transaction processing, in which master-files are renewed—a typical example is sales and inventory management system. DRESS is not as flexible as DEMOS in building an application system for end-users, while it has an effective file-accessing ability and a complete security for transaction data and user files. Computers of the same kind—DIPS (Dendenkoshu Information Processing System)—are used in both systems. However, different kinds of operating systems are used, according to differences in the systems' features.

For a long time, putting large-scale computers into common use, both have worked a lot in providing cheap and easily available information processing tools for users who couldn't afford to have their own computers. Recently, however, the role of both systems in that sense is rapidly becoming less than before, because of the drastic improvement in local processors (especially mini-computers and office computers) in regard to the efficiency and price.

From now on, NTT intends to provide various service menus for DRESS and DEMOS which cannot be sufficiently realized on a local-processing basis and will be fulfilled with on a remote processing basis. The service menus NTT has in mind are as follows: (a) hardware resources which local processors cannot be economically equipped with, such as:

- High-speed processing machine
- Mass storage system
- High-speed kanji-printer
- Large scale X-Y plotter;

(b) network linkage functions, such as:

- Network interface for various kinds of terminals
- Relay node for different kinds of computer systems
- Constructing efficient network for nation-wide data-processing systems;

(c) data distribution functions for database producers; (d) software package circulation functions for software authors; and (e) software production and debugging tools for various

kinds of computers and micro-processors, such as:

- Cross software library
- Software conversion tools
- Program generators
- Support tools for designing and documentation.

### COMPUTER COMMUNICATION REQUIREMENT IN NTT'S RCS

The NTT's RCS center facilities are decentralized into several districts (the NTT's RCS centers placement is shown in Figure 2) for the following reasons: (a) to match existing maintenance-and-operational organization for telecommunications networks; (b) to avoid expensive charges for the circuit use, normally determined according to the distances involved in the transmission; and (c) to safeguard against disasters, such as earthquake, fire and typhoon.

Lately, computer communications among these distributed centers or between them and other host computers have been required for the following reasons.

#### 1. End-user systems geographical range extension

There is a tendency for unifying the data-processings, which have previously been accomplished individually at

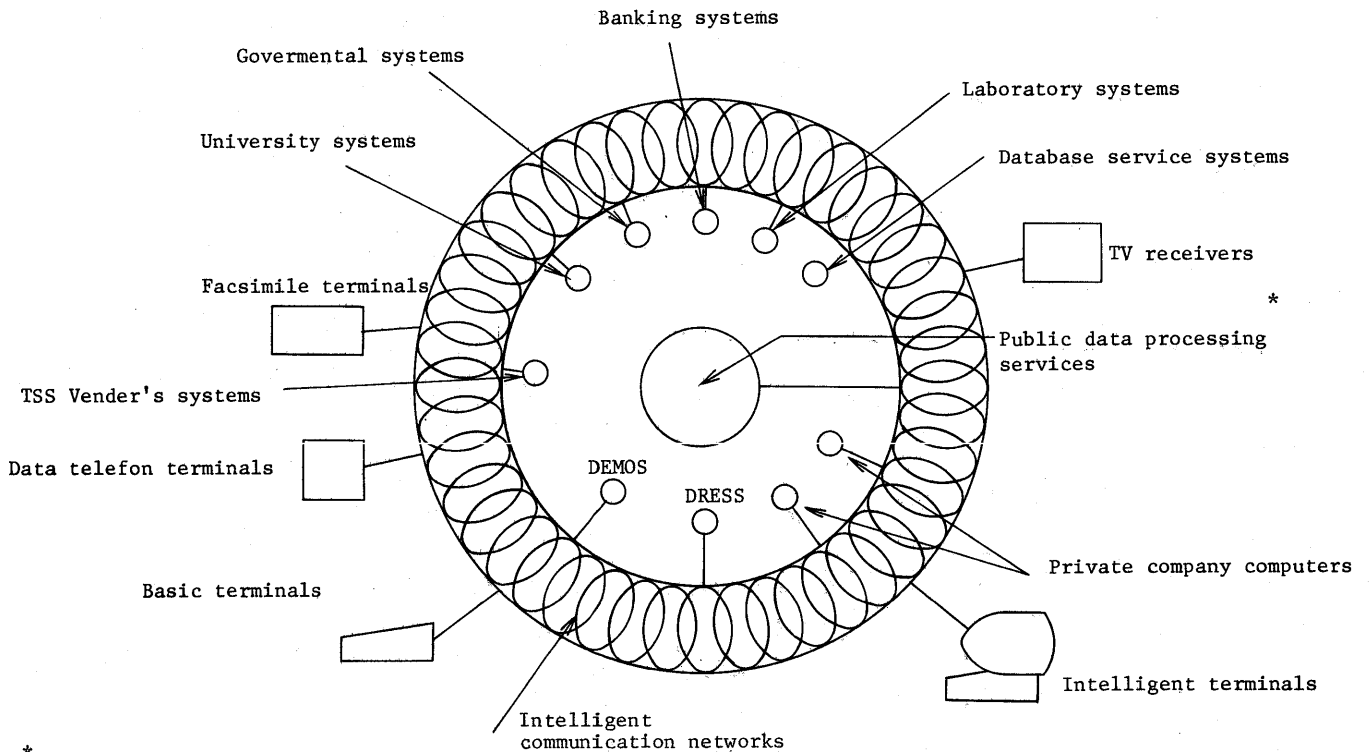
head offices and at branch offices or among different corporations so far. There is a requirement for constructing systems for sales management and inventory control, point of sales management, order entry, reservations etc. on a nationwide scale from the beginning.

There is a requirement to use the NTT's RCS network which already has many customers and sufficiently varied functions to afford a new opportunity for software-houses and database producers, for software circulation and data circulation.

#### 2. Functions distribution among centers within the same service

This is a means for sharing facilities at a center, which has specialized functions—the so called functional center—among general centers. NTT has already established in DEMOS the Extended Remote Batch Center equipped with high speeded CPU which is connected to all of the other centers by computer communications with high speed transmission line and high level protocols.

NTT intends to construct a large scale database center, a video information storage center etc. as a function center, believing that a total system can be developed economically and rationally using this approach.



\*  
 AUTOMATED METEOROLOGICAL DATA ACQUISITION SYSTEM  
 AGRICULTURAL INFORMATION DISTRIBUTING SYSTEM  
 EMERGENCY MEDICAL INFORMATION SYSTEM etc.

○: Data processing node

Figure 1—The network utility image.

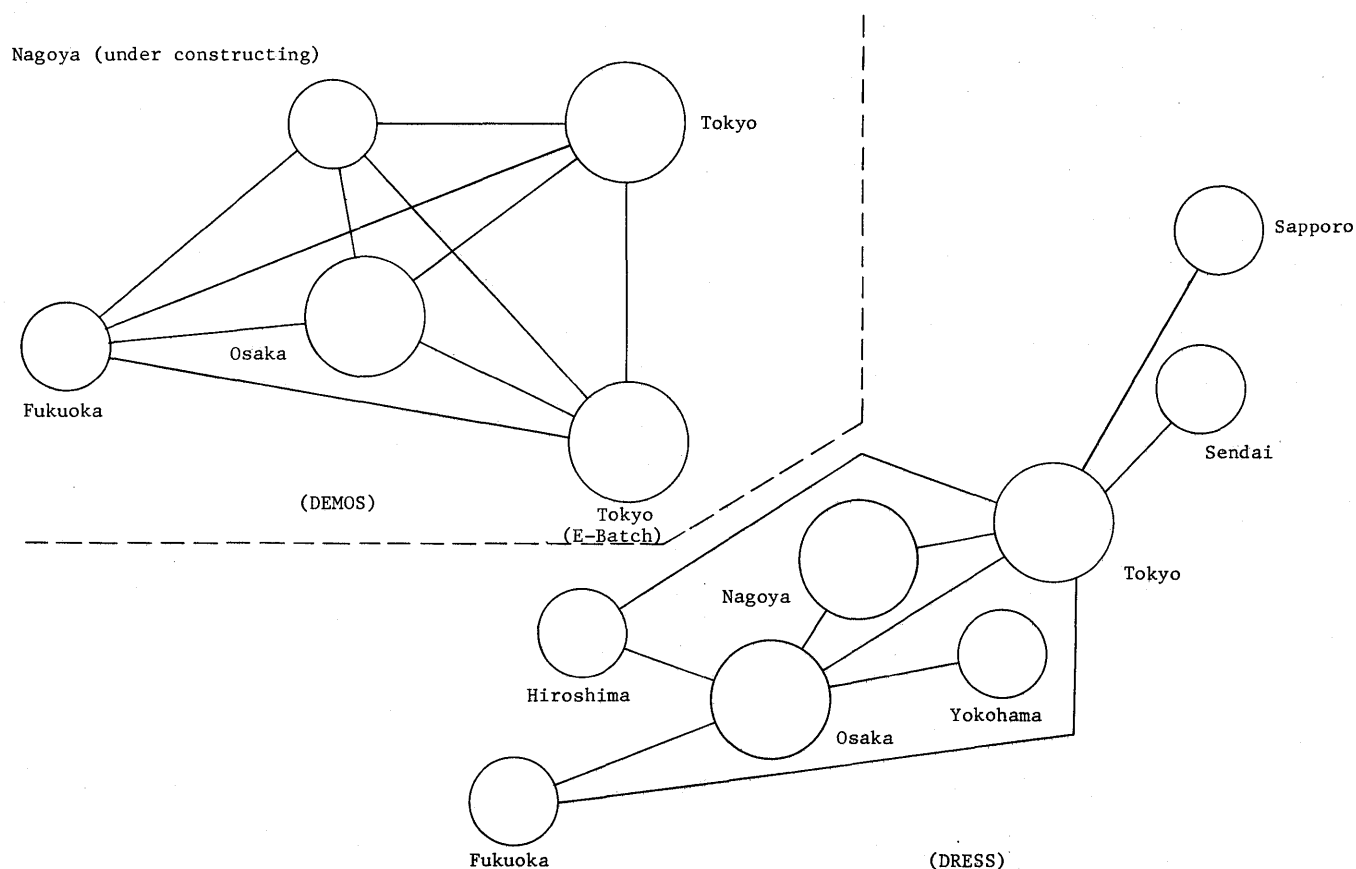


Figure 2—DRESS and DEMOS centers placement.

### 3. Customer need for DRESS and DEMOS unification

Lately, the requirement to realize a total system, by combining the merits of both DEMOS and DRESS, have been accelerated. A typical example is in regard to analyzing data in DEMOS which have been collected through the daily processing in DRESS. Though a method was considered by which to unify DRESS and DEMOS operating systems and to share the files in the same center in order to respond to these requirements, it was concluded that, for the time being, it is more favorable to regard DRESS and DEMOS as individual independent functional centers and to connect these centers by computer communications techniques.

### 4. Customer requirement for a distributed system connecting NTT RCS centers and non-NTT centers

A typical example of these requirements, which has emerged very clearly by now, is the connection with the database service vendor's center. Connection with the banking data-processing systems, governmental data-processing systems, non-NTT TSS centers, etc. will be seen in the near future. The following requirements for connecting with private company computer systems are appearing: (a) batch-processing the data in private computers which are

collected through the RCS network; (b) processing the data in private computers, making use of RCS program resources; (c) dealing with the overflow from private computers in the RCS; (d) employing either private computers or the RCS depending on the kind of work involved.

Items 1. and 2. concern only the same service. Item 3. concerns two services in different categories. Item 4. concerns different computer categories.

The NTT concept about basic computer communications functions, which should be realized to respond to these requirements and ways to implement them, are presented in the following.

### BASIC COMPUTER COMMUNICATIONS FUNCTIONS REQUIRED

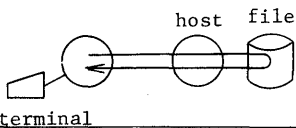
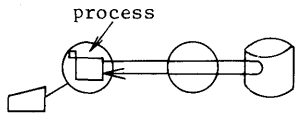
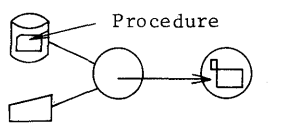
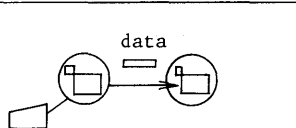
Basic computer communications functions include file transfer, file record access, job transfer and message transfer.

Table I shows the functions needed in NTT's services within the same services, among different services and among different computers. At the same time, it shows how far these functions have been realized as of now.

Some considerations about these fundamental functions are presented in the following.



TABLE I.—Functions Required in NTT's RCS

Function	Figure	Between Same category computers			Among different category computers	
		Within the same services		Between DEMOS and DRESS	Between DEMOS and other host	Between DRESS and other host
		Within DEMOS services	Within DRESS services			
File transfer		●	○	○	○	○
File record access			○	○		
Job transfer (includes job-step transfer)		●			○	○
Message transfer		●				

● : Already in service      ○ : Specifications already determined, being manufactured      ○ : to be developed in the near future

1. Both file transfer and file record access have their own application areas, depending on the volume of the data to be accessed. Of the two functions, file transfer was realized first, since its realization was technologically easiest. However, when retrieving database data from the remote host, etc., file record access is necessary anyway. Therefore, it is inevitable to support this function at the next stage. The cost (including processing charge, file charge and circuit charge) accounting result, depending on the data files placement and file record access method for an actual point of sales management system model, are shown in Figure 3.

In this model, distributed placement data files provide more efficiency, compared with centralized placement data files. If distributed placement is adopted, file record access is more efficient than the file transfer method.

2. When sharing program resources, job step transfer provides more efficiency than program file transfer does, when the program volume is sufficiently large. This is why the job step transfer method was developed in DEMOS.

3. Accessing the remote file records using the message transfer methods, the file access function can be easily substituted for without worries about locking and unlocking files, file backup etc. In this context, message transfer was realized in place of file record access in DEMOS.

4. Fundamentally, it is desirable that connection with the

remote resources should be transparent for users of the local host in distributed data-processing. For example, this transparency was realized in the following cases.

- The command 'ULIBRARY' (user library) sets the circulation software package into motion. In using this command, the users don't need to take into consideration at all in which host the program files exist. This is because the program files required are automatically transferred in the process within the ULIBRARY command, even if they exist in some other host.
- The NTT database management system—DORIS-2 (Dendenkosha's Online system for Retrieval of Information and Storage) has realized data retrieval from a remote host by applying the process-to-process data transfer method. Retrieval can be achieved without any considerations about the computer communications, by setting up in advance information about the files to be retrieved and the host in which they exist.

5. The full-scale distributed DBMS (Data Base Management System) is now under research.

In order to realize this DBMS, it would be necessary to prepare a basic function for distributed DBMS other than those discussed above.

## PROTOCOL IMPLEMENTATION

### *Protocols within DEMOS*

These protocols include file transfer, job transfer and message transfer. Protocol layer hierarchy is shown in Figure 4.

These protocols have been developed only to apply to DEMOS. These protocols were attained not only in order to meet customer needs, but also to study the technology required for developing the most generalized computer network architecture applicable to different computer categories—DCNA (Data Communications Network Architecture).

1. It has been possible to achieve very high level efficiency in these protocols, because the code systems, file structures, file identification methods and so on are the same within the same service.

However, the following points prevented the protocols from having the general applicability required to be extended for application to the communication field among the different computer categories: (a) separation of protocol layer hierarchy is not sufficient; (b) resources virtualization is not sufficient.

2. At the Host/Host protocol designing stage, every effort

was made to detach the fundamental functions from each protocol, so that those functions could be shared by each protocol in the form of protocol-commands.

As shown in Table II, 60 percent of the protocol commands used in a certain protocol are shared by other protocols.

Similarly, from the viewpoint of the number of program steps, the common use ratio is as high as 60 percent.

3. There would be two controlling methods for the process-to-process data transfer. One is the method in which only local process can control data transfer. The other is the method in which both local process and remote process have the right to control, on an equal basis.

However, for now, it is not necessary to realize this method at the risk of an increase in over-head time and at the risk of deadlock occurrence resulting from a right of control exchange.

4. A method has been adopted wherein the acknowledge response from the remote process cannot be acquired until the data transfer has entirely ended, even when a large volume of data is to be transferred. This is because it was concluded better to lay stress on transfer efficiency as a result of trading-off between recovery from difficulties and data transfer efficiency.

Difficulties in the center at the opposite end are almost the only ones to be recovered at the host/host level.

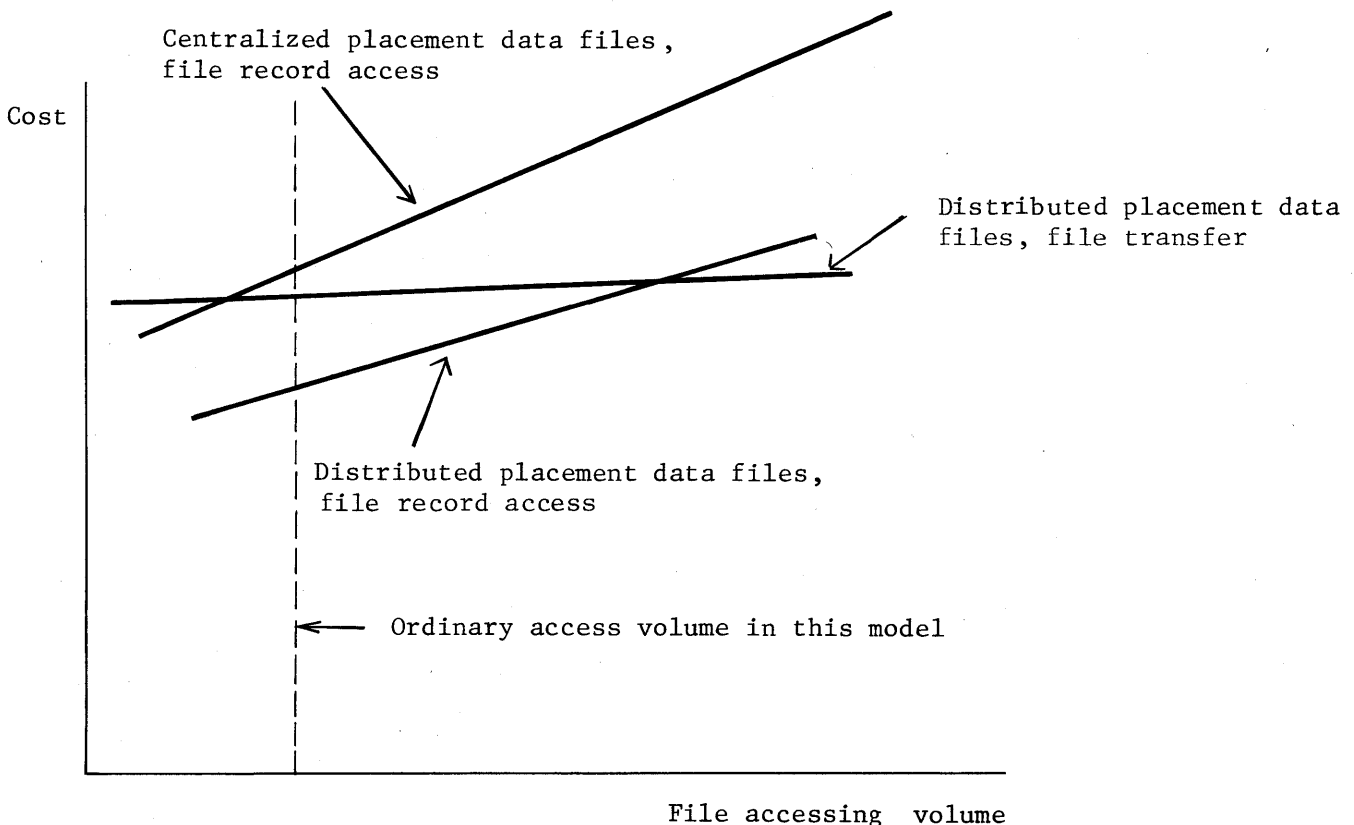


Figure 3—Cost accounting result for actual point of sales model.

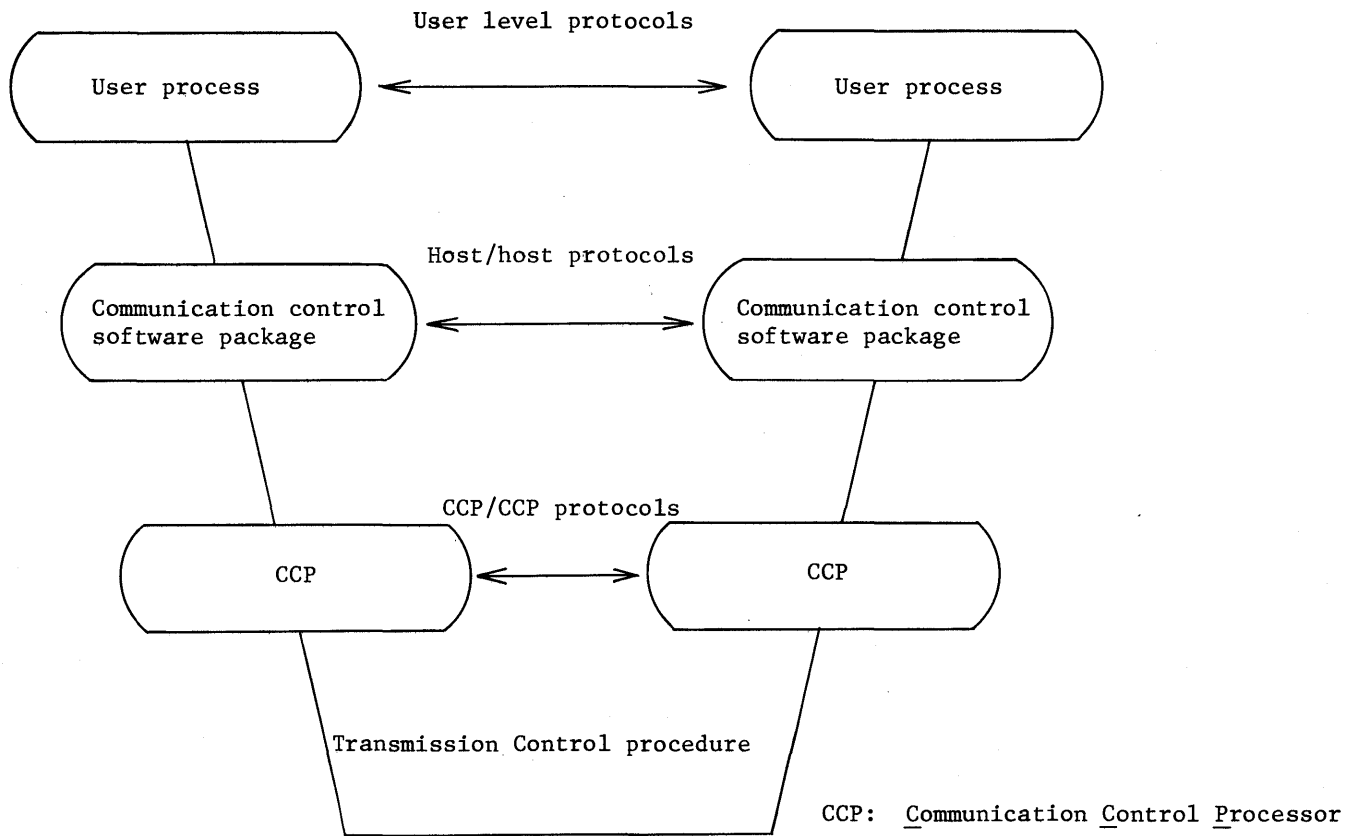


Figure 4—Protocol layer hierarchy.

TABLE II.—Protocol Commands Common Use Ratio

	Commands	Steps
Protocol commands and their steps	27.0	33.6 *
Total protocol commands and their steps	72.0	100.0 *
Average values per protocol	5.5	7.7 *
Average commands used in common by other protocols and their total steps per protocol	3.4	5.1 *
Common use ratio [ (4)/(3) x 100% ]	62.0 %	66.0 %

Number of protocols: 13

\* The number of steps is not a real one. It is shown in the form of the ratio when compared with the number of steps in the total protocol commands (=100.0).

However, the difficulties probability in NTT centers is very small and, once difficulties arise, the connection must be cut off in most cases where troubles are not recovered within a permissible time lapse.

*Protocols within DRESS and protocols between DEMOS and DRESS (file transfer protocols)*

These protocols are the first protocols implemented based upon DCNA on a full scale.

DCNA has strong generality and is applicable to various computer categories.

NTT has been developing this architecture as a common carrier's duty, in cooperation with several computer makers. DCNA protocol layer heirarchy is shown in Figure 5.

The implementation characteristics for these protocols are as follows:

1. As the internal forms of the user management and the file management etc. were different between DRESS and DEMOS, the mapping method between these forms was the most important subject.

2. As these protocols are tentatively implemented by using the existing communication control software package in the first place in order to meet increasing user demands, these protocols are not entirely based upon DCNA. The protocols which are fully based upon DCNA will be created

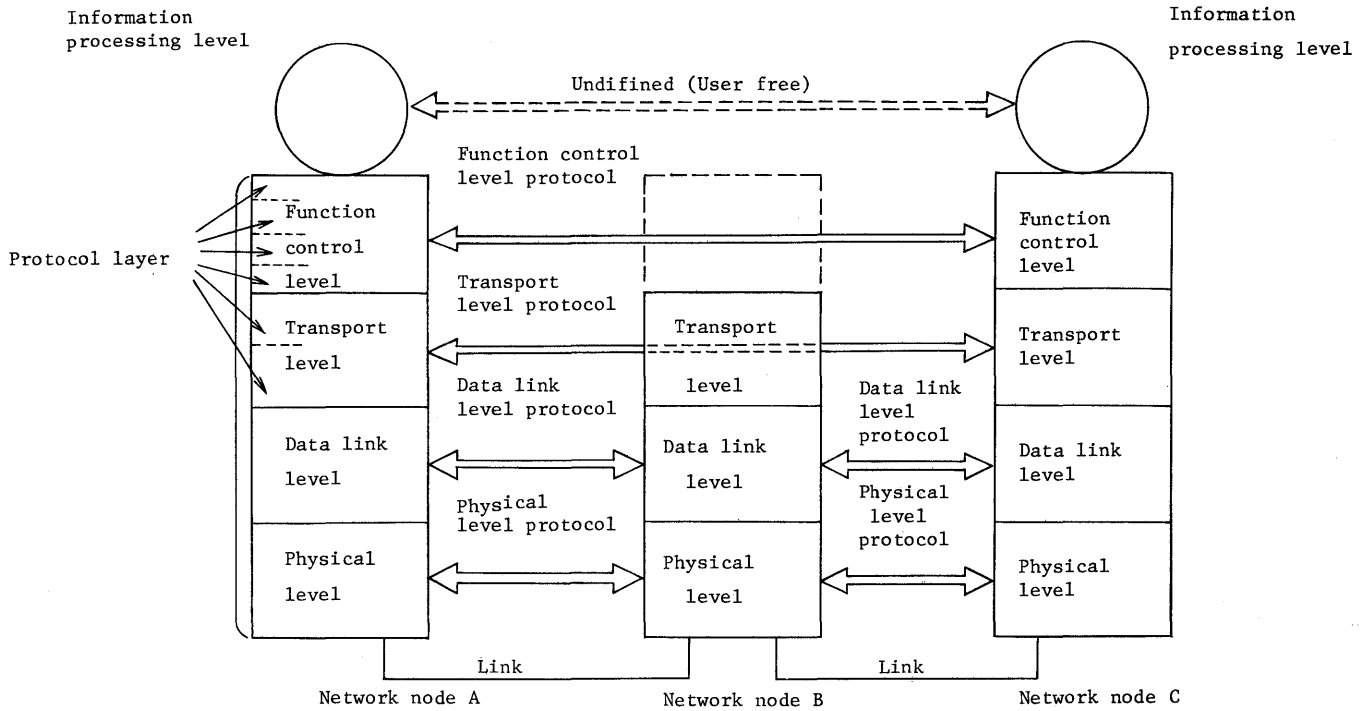
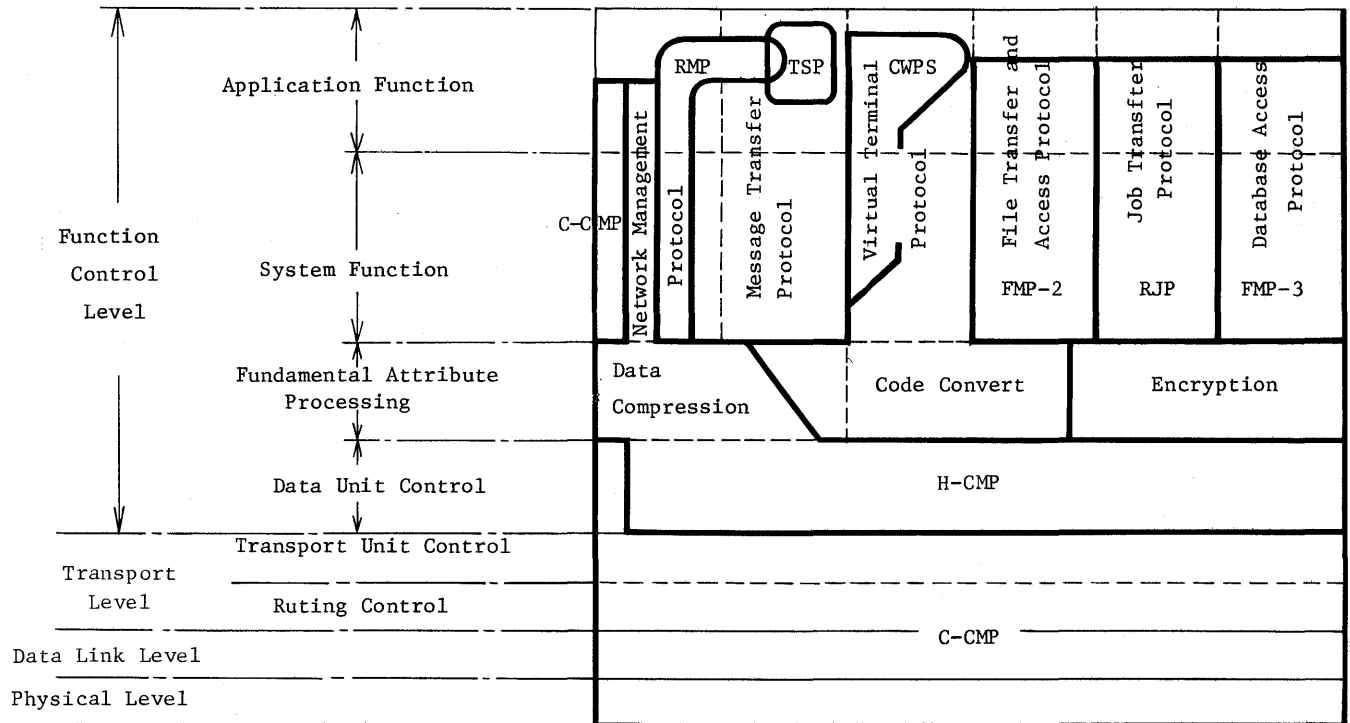


Figure 5—DCNA protocol layer hierarchy.



DCNA Software Products

- C-CMP : CCP-Communication Management Program
- H-CMP : Host-Communication Management Program
- RMP : Remote Maintenance Program

- TSP : Time Sharing Package
- CWPS : Communication Word Processing System
- FMP : File Management Program
- RJP : Remote Job Entry Package

Figure 6—DCNA installation image.

only after the new communication software package has been produced.

It is true that the shift from the old protocols to the new ones can't be accomplished at the same time in all centers. Therefore, the old protocols and new ones will be forced to co-exist with each other temporarily.

In order to solve this problem, a converter is being developed which would act as the gate-way between these protocols.

3. It has been decided that the control-management of the whole network should be accomplished independently by each center. The reason is to match the existing maintenance and operation method in the centers and to avoid the risk of cutting off the computer communications among all centers when the network control management center fails.

## CONCLUSION

The most important subject to solve now is how to communicate among different kinds of computers.

The present NTT schedule for this is as follows:

1979: Communications between NTT computers and other computers have been realized on the host-to-basic terminal-connection basis (communications at the data link level or the transportation level).

1981: File transfer protocols implemented in DRESS and DEMOS will be opened to the public.

1983: Communications between NTT computers and the products of the computer makers who co-operate with NTT will be established on a full DCNA basis. A DCNA installation outline is shown in Figure 6.

After 1983: The converter which realizes the communications between NTT computers and other computers, on the basis of architecture other than DCNA, will be provided. At this stage, the NTT RCS will act as a relay node among computers as well as a general purpose processing node.

While the communication among different kinds of computers will greatly contribute to the computerization, it is sure that there will be some unforeseen problems. For example, possible problems include data and message security, privacy protection and increase of influence upon society produced by the difficulties of a certain node because of the increase of the bilateral dependence among computerized systems.

As discussed above, NTT's RCS will play the most important part as an infrastructure in Japan.

Therefore, sufficient assessment of these problems must be attained when developing the computer communication techniques.

## REFERENCES

1. Yutaka Ohshima and Yutaka Nakagawa, etc., "Function and Mechanism of Host/Host High-Level Protocol," *Review of the Electrical Communication Laboratories*, Vol. 28, No. 3, 4, 1980.
2. Seiichi Kawazu and Kohhei Ohnuma, etc., "Distributed Database Application to Decision Support Systems for Chain Stores," NTT Yokosuka Telecommunications Laboratory, May 1979.
3. Kaoru Kubo, "DEMOS-E Network," *Information Processing*, Vol. 20, No. 4, 1979.
4. Akira Takai, Ikuo Ohsawa, and Hiroshi Saya, "General Purpose Time-Sharing Service Improved by New OS," *Japan Telecommunications Review*, Vol. 20, No. 2, April 1978.

# Local area data distribution

by THOMAS G. ALBRIGHT and ROBERT J. WALLACE

Printer Terminal Communications Corporation  
Ramona, California

## LOCAL AREA DATA DISTRIBUTION

The Information Processing Industry now has available a new link-technology between information and the information user. The new technology is the local radio frequency (RF) voice-grade data broadcast channel.

Radio frequency communications have been commonly used in data communications networks in the past. However, their use has been predominantly restricted to wideband channels, specifically, with point-to-point surface data communications utilizing microwave transmission, and with point-to-point satellite data communications.

For the purpose of discussion, we will refer to this new link-technology as LADD (Local Area Data Distribution). LADD, like microwave and satellite RF communications, is not a data communications network in itself, but a system element. As a system element, it is available for use by all data communications network operators who find it useful.

One of the beneficial qualities of LADD is that it is not a leading edge technology which needs refining to be reliable, predictable, and cost effective. Rather, it is simply the union of existing broadcast radio techniques with digital modulation-demodulation (Modem) techniques to achieve digital broadcasting. However, one of the difficulties in comparing LADD to other data communication technologies on a detailed technological level is the lack of extensive research experience in the LADD service frequency range and bandwidth. A summary of the most relevant data communications research experience in the LADD service frequency range, the ALOHA project, may be found in Binder *et al.*\* Yet, even the ALOHA project provides a poor reference point due to its broad bandwidth and half duplex service mode. These two differences have led ALOHA-related studies toward optimizing scheduling protocols rather than expanding the basic knowledge of the RF medium as a simplex data channel, particularly using voice grade service frequencies. Although it shares RF communications methods with satellite and microwave communications, LADD is much different in capabilities. Table I illustrates some of these differences.

\* Binder, R., N. Abramson, F. F. Kuo, A. Okinaka and D. Wax, 'ALOHA packet broadcasting—a retrospect', *Proceedings, National Computer Conference (1975)*, page 203.

Because of these differences and LADD's capabilities, LADD has specific and limited applications capabilities. The LADD system's configuration includes several components:

*The physical link* connects the controlling site processor which interfaces the user's host computer or network to the LADD Broadcast System. Normally this link is a conditioned leased line utilizing BSC protocol. More sophisticated means of implementing error detection and correction on this link are possible and compatible with LADD configuration. The capability of a dial back-up of this physical link is built in to the LADD system to improve systems availability.

*The controlling site processor* is the gateway from the user's network for data messages from the network that are to be distributed. It regulates the flow of addressed data messages, received from the user's host computer or network, according to pre-defined user priorities, to the LADD Broadcast System. The net bandwidth capabilities of the controlling site processor over the physical link are tuned to the maximum net bandwidth of the LADD Broadcast System in order to simplify control. The controlling site system, therefore, acts as an arbitration and store-and-forward device. In addition to these duties, it can: (1) selectively invoke special data message handling requirements on a message-by-message basis, (2) cause the transmission of a predetermined test message for systems checkout and maintenance; and (3) record user accounting information for later analysis.

*The LADD Broadcast System* exists at the FCC licensed Broadcast Service Operator's facility. It is linked to the data communications network controlling site processor via the physical link. The LADD Broadcast System provides a regulated signal to the Broadcast Service Operator to be injected into the RF broadcast transmission stream.

The functions of the Broadcast System are:

- Maintain the physical link protocol to the controlling site processor.
- Provide the broadcast signal of data messages which includes encapsulation of data messages according to LADD protocol (see Appendix A—LADD Transmitted Data Organization); multiplexing messages for optimum use of the broadcast channel, and modulation of messages to comply with RF Broadcast Service requirements.

TABLE I.—Comparison of RF Communications Methods

TECHNOLOGY:	LADD	SATELLITE	MICROWAVE
FACTOR:			
Transmission Mode	broadcast or point-to-point	point-to- point	point-to- point
Reception	voice-grade to	wideband	wideband
Channel Bandwidth	wideband		
Cost of user-site receiver/modem	low	high	high
Range (physical or regulated)	short	long	short
Practical Transmission Direction	one-way outbound	two-way	two-way

- Provide selective data message special handling capabilities. Data messages from the controlling site may be "tagged" to indicate such special handling as encryption, forward error correction (FEC), and time and date stamping.
- Collect accounting data on the utilization of LADD.
- Provide feedback to the controlling site on the status of the LADD Broadcast System and the FCC licensed Broadcast Service.

The FCC licensed Broadcast Service Operator simply accepts the signal from the LADD Broadcast System, injects the signal into the transmission stream, and transmits the signal as part of his normal operations.

The LADD Antenna-Receiver-Terminal is the final component handling the flow of data messages within LADD. The Antenna receives the RF signal and delivers it by cable to the Receiver. The Antenna is usually mounted outdoors. When the multiple Terminals are used within a single facility, only a single Antenna is required. The Receiver, upon receiving the RF signal from the Antenna, demodulates the RF signal to a digital and demultiplexes the signal, preserving only the data message sent to its related Terminal. In addition, the Receiver performs necessary receiving-end functions required by selective special data message handling. The Terminal disposes of the digital signal in accordance with the wishes of the user.

The broadcast capabilities of the FCC licensed Broadcast Service Operator are key elements in the physical organization of LADD with respect to determining applications capabilities. Appendix B, FCC Licensed Services Approved for Data, summarizes the major restraints and capabilities of transmission services capable of supporting LADD technology. The three major applications-related factors are:

The channel bandwidth authorized for data. This is the maximum allowable bandwidth according to FCC regulation.

It is important to note that the user will be sold considerably less bandwidth by the FCC licensed Broadcast Service Operator. The Broadcast Service Operator, in order to protect his operation from violating FCC regulations (or affecting his main channel, where one exists), will provide his own guardbands. While these guardbands protect the Broadcast Service Operator, they reduce the user's bandwidth as actually delivered.

The transmission mode authorized by the FCC for that particular transmission service. The two available modes are broadcast and point-to-point. The difference between broadcast and point-to-point mode is contrasted in Davies *et al*\*\* when introducing Packet Broadcast Systems. The LADD RF data channel inherently has broadcast mode capabilities as a result of the FCC licensed Broadcast Services medium. Point-to-point mode communications are a result of the channel protocol's ability to selectively address individual terminals. Even when broadcast mode is inherent and point-to-point mode is enabled by channel protocol, the ability to use one mode or the other, within the law, is regulated by the FCC. If a transmission service is not authorized for both modes of transmission, it is implicit that it can only be used for one. The extreme flexibility of addressing provided by RF communications makes it difficult (and undesirable) to police occasional deviations by the Broadcast Service Operator with regard to adhering to the authorized transmission mode.

The de facto means utilized by the FCC in policing transmission mode violations is the intent of the user. Therefore, if an application under consideration is point-to-point in nature, a transmission service should be selected which is authorized for point-to-point. Similarly, a primarily broadcast application should select a broadcast mode authorized transmission service. To be safe, if the primary mode of applications traffic is uncertain, a transmission service should be selected which is authorized for both broadcast and point-to-point mode transmissions.

The normal reception range of the transmission service. A LADD message is transmitted in all directions simultaneously. All Antenna-Receiver-Terminals within reception range which are tuned to its frequency receive the message virtually at once. The normal reception range is therefore the key to the geographic coverage achievable with a single transmission system.

Local geographic and man-made features can affect the actual reception range as can the height and direction of the Antenna. Because of these factors, actual reception range is more properly measured in signal strength (intensity) at the Receiver. Signal strength must be measured at all Antenna-Receiver-Terminal sites as part of a site survey prior to implementing LADD. If the signal strength is not great enough, the reliability of the Receiver output will be suspect (yes, Garbage-In, Garbage-Out applies to RF Systems too!). The maximum Effective Radiated Power (ERP) of the Broadcast Service Operator's equipment is regulated by the FCC and will affect the normal reception range.

\*\* Davies, D. W., D. L. A. Barber, W. L. Price, and C. M. Solomonides, *Computer Networks and Their Protocols* (1979), page 155.

It is well at this point to summarize what LADD is, and what it is not:

LADD is not leading edge technology but is a practical union of RF communication and Data Modem techniques. It is a data communications systems element but is not a data communications network. It requires no FCC license for the user because transmission capability can be purchased from Broadcast Service Operators. It requires a controlling site processor in order to interface with a data communications network. It can transmit in broadcast mode as well as point-to-point mode. It is one-way in operation with outbound transmissions. It has a useful range of up to 100 miles from the transmitter. It has primarily voice-grade (300-9600 baud) net transfer rate capabilities.

With the LADD system described, one has only to review a few trends and events of the 1970's in the areas of Information Processing in order to appreciate how such a capability could fit into data communications network designs.

New service competitors of Bell System services have emerged and are focusing on the data communications market.

- Leased Line Services face lower priced competition in all grades: sub-voices, voice, and wideband.
- Hybrid Services competitors have emerged, providing improved services at lower prices.
- Switched Services have no significant new competition.

The Bell System changed from a uniform rate structure (pre-1974) to a Hi-Lo density rate structure (mid-1974 to mid-1976), and then to MPL rate structure, resulting in sharp decreases in long haul rates and sharp increases in short haul rates.

Decreasing computer prices have stimulated growth in the number of computer sites, especially for mini-computers.

The microprocessor has become a standard element in most terminal equipment, giving terminals computational capability.

The demand for data communications, hardware, staff, and services continues to grow at a high rate.

In summary, more installed computing capability, increasingly dispersed computing capability, and improved Leased

TABLE II.—Remote Terminals Attributes

Application Attribute	Application More Desirable	Application Less Desirable
A. Number	many	few
B. Dispersion	most within the range of a single broadcast transmission system	some within the range of a single broadcast transmission system
C. Throughput	high	low
D. Movements and changes	frequent	infrequent
E. Installation delay tolerance	must be within a few days	can sustain 6 weeks, 8 weeks or longer

TABLE III.—Message Traffic Attributes

Application Attribute	Application More Desirable	Application Less Desirable
A. Volume	high	low
B. Proportion common to multiple remote terminals	high (broadcast mode) low (point-to-point mode needed)	
C. Patterns of flow	erratic	predictable
D. Maximum instantaneous bandwidth required	2400 baud	2400 baud
E. Priority levels	many	few
F. Immediate delivery	required for some of the volume	not required
G. Delivery within 24 hours	required (or tolerable) for some of the volume	not required (or tolerable)
H. Security (encryption)	required on a selective message basis	not required
I. Transmission error detection	required	not required
J. Transmission error detection and correction (FEC)	required on a selective message basis	not required

Line long haul pricing rates have created a need for local (short haul) data distribution. While LADD cannot satisfy all of this need because of its specific capabilities, it can help satisfy some of it.

Analysis of the LADD capabilities measured against the general attributes of remote terminals and message traffic within a data communications network yields a means of measuring applicability. Attributes which, when analyzed, measure the applicability of LADD as a systems element of data communications networks, are shown in Tables II and III.

Another way to estimate the applicability of LADD is to analyze it relative to data communications network design constraints. A point by point review produces a general view of LADD specific enough to determine if the technology can be beneficially added to an existing data communications network. The following paragraphs are just such a review based on the design constraints identified by Dixon R. Doll† regarding network design preliminary information:

*Number and Locations of Processing Sites:* A key factor in implementing Local Area Data Distribution (LADD) is that there be a single controlling site regulating the flow of data traffic to be distributed from the processing sites. Therefore, the number and locations of processing sites in a data communications network is inconsequential to the ability to implement LADD. Both centralized and distributed data

† Doll, Dixon R., *Data Communications—Facilities, Networks, and Systems Design*, 1978, p. 4.



communications networks will find LADD applicable to their operations.

*Number and Locations of Remote Terminals:* The locations of remote terminals is a key design parameter for implementing LADD. All remote terminals located within the range of the Broadcast Service will have essentially equal ability to receive data messages. The number of remote terminals within the broadcast range is not a parameter affecting the physical implementation of LADD.

*Information Flow Patterns Between Terminals and Processing Sites:* Information flow using LADD is, by definition, outbound only from the processing sites through the controlling site to the remote terminals. Erratic patterns of information flow which would make it impossible to economically install a leased line network to distribute data have no effect on LADD's ability to distribute data.

LADD can function as the outbound channel in two-way communications, wherein the inbound and outbound data each have dedicated channels. In this way, for example, high volume printing on multidrop networks can be sent via LADD, thereby maintaining low response times for terminals.

*Types of Transactions to be Processed:* Transactions of any type can be distributed using LADD. One of the benefits of LADD's being able to operate in broadcast mode as well as point-to-point mode is the additional transaction types which can be handled. An example of such additional transaction types is outbound policy information such as price changes. This kind of information is usually composed centrally, voluminous in nature, and delivered by mail. Virtually instantaneous broadcast of such critical and detailed information to all receiving sites concurrently cannot be achieved with any other commonly available data link technology.

*Traffic Volumes for Transaction Types:* Traffic cannot achieve a net transfer rate in excess of the net transfer rate of the broadcast channel. This value is dependent on the Broadcast Service. Traffic volume is a factor in judging the economy of employing LADD technology. The greater the volume of traffic addressing multiple remote terminals, the lower the cost per message delivered. Increasing the volume of traffic addressing individual remote terminals increases the cost per message delivered.

*Urgency of Information to be Transmitted:* Transmission of data to remote terminals is practically instant, up to the maximum net transfer rate of the broadcast channel. When the volume of data to be transmitted at any instant exceeds the maximum transfer rate of the broadcast channel, the controlling site is called upon to arbitrate the message sequence priority for transmission. A mix of data messages of several degrees of priority produces the best economical performance of a LADD system by spreading the load evenly.

*Capacity Reserved For Traffic Growth:* The nature and volume of excess capacity is measured in terms of the maximum transfer rate of the broadcast channel and the patterns and priority of message traffic. A LADD system will have 24 hours a day availability for use under normal conditions. Generally, a LADD application will have capacity in reserve of twice the prime time capacity.

*Acceptable Undetected Information Error Rates:* The information bit error rate of a properly tuned LADD is less than  $10^{-7}$ . This rate can be improved by utilizing link forward error correction (FEC) routines. Error checking and correction routines compromise throughput for improved accuracy by reducing the maximum transfer rate of the broadcast channel by their overhead. In many LADD implementations, error checking and correction may not be necessary because the technology is intrinsically superior to common carrier links whose bit error rates, according to Doll<sup>‡</sup>, range between  $5 \times 10^{-5}$  and  $5 \times 10^{-6}$ .

*Reliability and Availability:* The reliability of a LADD system will be directly related to the reliability of the controlling site system, the physical link, the LADD Broadcast System, the FCC licensed Broadcast Service, and the remote Antenna-Receiver-Terminals. The Broadcast Service is not only exceptionally reliable by information processing standards, but also usually has a full time on-site engineering staff to correct failures. Certain components of the system such as the physical link and the LADD Broadcast System can be implemented redundantly where improved reliability is required. The availability of the system is generally 24 hours per day less down time due to equipment and power failures.

*Availability of Financial Resources:* Most data communications networks with the need for LADD capabilities already have a controlling site system in place. With this environment, adding Local Area Data Distribution capability to the network would include the costs for the LADD Broadcast System and Antenna-Receiver-Terminals hardware purchase and their installation, plus a monthly service charge for the physical link and for the FCC licensed Broadcast Service, and maintenance charges. The most exciting aspect of LADD for the financial decision-maker is that when utilized effectively it has the ability to make the cost of distributing information less than the cost of the paper it's printed on.

Because LADD is new, and specific and limited in the applications in which it is useful, it will not be a commonly used link-technology for several years. Most probably, the initial demand will continue to come from large corporate users whose needs are a perfect fit to LADD's capabilities. Additional growth in LADD's usage will come as a surge when the value-added common carriers integrate LADD's capabilities into their hybrid networks.

Ultimately, LADD link technology will be just another data communications network building tool and in common use just as satellite and microwave RF communications are now becoming.

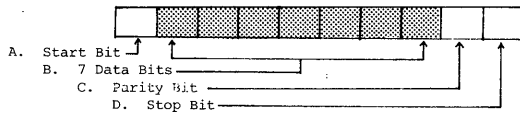
## APPENDIX A

### LADD TRANSMITTED DATA ORGANIZATION

#### I. Transmission Mode Asynchronous

<sup>‡</sup> Doll, Dixon R., *Data Communications—Facilities, Networks, and Systems Design*, 1978, p. 4.

## II. Character Format



## III. Message Block Organization

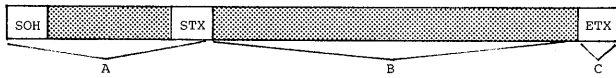


Figure 1

### A. Message Header—

1. SOH—Start of header USASCII control character
2. Header control characters:
  - a. Primary addresses—4 printable USASCII characters
  - b. Secondary addresses (optional)—up to 7 additional addresses of 4 printable USASCII characters undelimited by other characters
  - c. Daily message sequence number (optional)
  - d. Time and date stamp request (optional)
  - e. Selective Forward Error Correction enable (optional)
  - f. Selective encryption enable (optional)

### B. Message Text—

Message Text may include any ASCII character with the exception of ETX. Messages may be of any length; however, invocation of certain selective message special handling features may require that messages not exceed some specific length.

### C. Message Trailer—

ETX—End of text USASCII character

## IV. Protocol Overhead

- A. Characters contain 70 percent data bits yielding 30 percent of the bits transmitted per character as protocol overhead.
- B. Message blocks contain a minimum of 7 characters of protocol overhead if no optional header control features contribute to protocol overhead on an elective basis and their overhead factors are to be considered as part of the cost of utilizing the respective features, not as part of the general protocol overhead. Basic message block overhead therefore becomes a function of message text length as illustrated by the following chart:

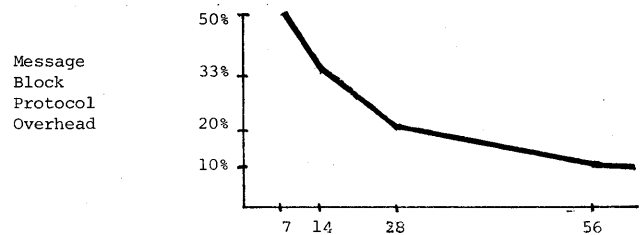


Figure 2—Message text length (characters)

At a message text length of 1000 characters the message block overhead is approximately 1 percent.

- C. Net protocol overhead is the total of character protocol overhead plus message block protocol overhead as follows:

Character protocol overhead	= 30%
Message block protocol overhead*	= 3.5%
(remaining 70% × 5%)	
<b>Total Protocol Overhead</b>	<b>— 33.5%</b>

\* Note—for the purpose of quoting a definitive overhead value message text length was assumed to be 56 characters.

## APPENDIX B

## FCC LICENSED SERVICES APPROVED FOR DATA

<i>Name of Service</i>	<i>Transmission Mode</i>	<i>Defining FCC Regulations</i>	<i>Service Frequency Range</i>	<i>Bandwidth Authorized for Data</i>	<i>Normal Reception Range</i>
Subsidiary Communications Authorization (SCA)	Broadcast	73.310	88-108 MHZ	22 KHZ (assumes stereo main channel)	100 miles (class C station)
Radio Common Carrier (RCC)	Point-to-Point	11.509 (3), (1) & (2)	158 & 454 MHZ	3 KHZ	Up to 60 miles
Domestic Public Land Mobile Radio Service (DPLMRS)	Broadcast and Point-to-Point	90.207	470-512 MHZ	3 KHZ	Up to 80 miles
Multipoint Distribution Service (MDS)	Broadcast and Point-to-Point	21.903 (a) & (b)	2150-2162 MHZ	6 MHZ	Line of Sight

## NOTES

- I. The means used by the FCC in determining the nature of a communications application is the user's intent:
  - A. Broadcast alone implies not point-to-point and usually implies public service.
  - B. Point-to-point alone implies not broadcast.
- II. The user is not required to obtain FCC licensing because the transmission service operator has already done so.
- III. Pricing on some of the transmission services is subject to Federal and State tariffs.

## Overview of the Computer Architecture Area

In the architecture area, the following six important topics are included: super-computer systems, data base machine and issues on database management systems (DBMS) standards, intelligent memory, architecture for local area networks, network data access support technology, and survivability criterion for the distributed data processing (DPP) networks.

A number of problems in the science and technology field require enormous computational power. The systems that are capable of solving these problems are called supersystems. Dynamic architecture is an effective way to provide a source of computer throughput. They can be classified into two areas: (1) adaptation of hardware resources on instruction and data parallelism, and (2) reconfiguration of hardware resources into different types of architecture—array, pipeline, multicomputer, multiprocessor. Two papers are presented on this subject.

Unlike the supersystem, the data base machine is a new and an important computer architecture. It has different characteristics as compared to the conventional number crunching systems and the supersystem. In the data base machine and standard issue session, a single joint paper will be presented by the authors and will be followed by an in-depth discussion on the issues of DBMS standards. The session will assess the progress made in the data base machines area, determine the functional capabilities and limitations of the present data base machines, and examine the issues on DBMS architecture, data models, and data languages from the point of view of present and future data base machines.

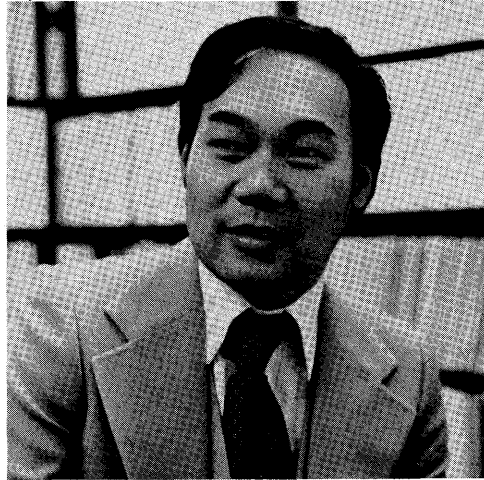
Memory systems play an important role in computer system performance. With the advent of LSI and VLSI technologies, it becomes technically and economically feasible to integrate logic and memory together—intelligent memory. Such memory systems will have high impact in future computer architectures. Application of intelligent memory to radar tracking applications, context addressible memory, and conflict free memory are discussed in that session.

With the advent of computer communication technology, interest is growing in loosely coupled computer system architectures in which the computing and mass storage components are connected via a high speed local network. In such a system, the computers can operate autonomously, as in a "distributed processing" approach, yet can share files and communicate among themselves at extremely high data rates. That session explores in detail one design for such a loosely coupled network system. This will permit the various tradeoffs involved in the design to be discussed in depth. A discussant will contrast the approach described and the decisions made with other alternatives.

Data access of a distributed data base system is one of the important areas that has been of interest to many researchers as well as practitioners. The problem is complicated by the fact that the data model, the host processor, the view of the data, etc. may be different from one site to another. Issues which will be discussed are data transfer vs. data distribution and cost/performance of different data translation models.

A distributed data processing (DDP) system is made up of telecommunication links between computers installed at network nodes. Interacting factors which affect DDP survivability include: data set and program distribution across nodes, network architecture, link and node redundancy, and the number of nodes and links in the network. The criterion for DDP survivability will be examined in terms of these factors and their interactions.

Because of the diversified nature and widespread interest of computer architecture, there are other sessions in the conference that are related to computer architecture. The participants interested in this subject area should also consult other sessions in this conference.



Wesley Chu  
*Area Director*



# The Control Data loosely coupled network lower level protocols

by WILLIAM C. HOHN

Control Data Corporation

Arden Hills, Minnesota

## INTRODUCTION

The Control Data Loosely Coupled Network (LCN) provides a form of two-party cooperative communications, similar to that described by Enslow (1). Put another way, LCN acts as an agent for Multi-Mainframing and distributed processing by providing a means for interconnection of, and information exchange amongst, a collection of mainframes or mainframes and peripherals (all referred to as "hosts" in the remainder of this paper).

LCN appears in a host as that software which provides network services to applications. These services include permanent file transfer, queued file transfer, application to (remote) application chat-chat, and shared rotating mass storage.

An equipment called a Network Access Device (NAD), attached to a host using the channel protocol native to that host, is the hardware entry point to the network. NADs in turn are interconnected by bit-serial trunks. Up to 32 NADs can be attached to a trunk; up to four trunks may be attached to a NAD. The combination of NADs and trunks provide the interconnections between hosts, and thus the hardware path for information exchange amongst hosts.

A set of protocols define and control LCN activities. They are shown, in Figure 1, as they relate to the ISO Open System Interconnection Reference model (2). Levels read down the page from the highest (application) to the lowest (physical). The remainder of this paper is limited to the protocols of the lower four levels.

## VIRTUAL CHANNEL

Conventionally two hosts have been coupled in one of three ways,

- a) common memory storage
- b) channel-to-channel (a variation is shared disk)
- c) communication lines

in order of descending performance. In addition, operating systems tend to be cognizant of these variations since data rates and response times vary dramatically with the form of coupling. What differentiates LCN from the traditional in-

terconnect schemes are the characteristics of the interconnect.

A host NAD, which interfaces a host to the remainder of LCN, is designed to the channel specification native to the host—including the maximum data rate of the channel.

NADs in turn are interconnected by shared trunks. The amount of trunk bandwidth available to a given NAD to NAD transfer is dependent on the loading of the trunk, varying from a maximum of 50 megabits (minus overhead) down to some minimum but nonzero value as trunk loading increases. Loading refers to how many NADs are attempting to use the trunk simultaneously.

From an operating systems point of view, a NAD to NAD transfer can occur at maximum channel rate, or at some lesser rate down into the realm of communication lines. This effect is analogous to virtual memory in that as the number of jobs mapped onto real memory exceeds the size of real memory, the execution time for each job increases.

A second parallel to virtual memory is that NAD to NAD transfers replace what is otherwise a system bottleneck with a slow transition. For example, coupling synchronous transfer rates of different value. The analogy in virtual memory is the mapping of a 10 million word program onto a half million word real memory without reprogramming for overlays or special I/O techniques.

In summary, LCN appears as neither a channel extension nor as a communications scheme, but rather as a virtual channel.

## PROTOCOL DESIGN CONSIDERATIONS

A design decision was made that the NAD should incorporate the lower protocol levels up to and including the transport level, primarily because of the problems associated with

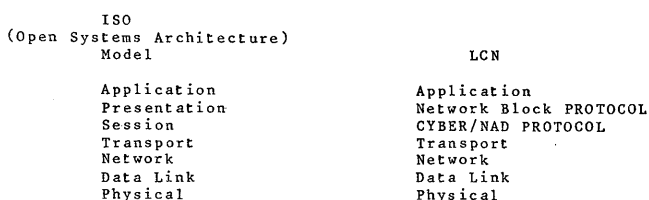


Figure 1—Protocol levels.

resource contention. Other factors influenced by and therefore favoring this decision included error recovery, performance of the network, simplicity of design, commonality throughout the network, and integration with existing hardware. Inspection of these factors with greater resolution revealed that the implementation had to:

- a) meet design goals at low cost
  - connectivity # units, distance, data rates
  - accessibility controlled access, multi-path
  - performance overhead
  - dependability detect errors, graceful degradation
  - maintainability fault trace
- b) Which translate into givens
  - serial trunk low cost
  - buffering data rate matching
  - communication like messages vs ready/protocol resume
  - intelligence to manage it all
- c) and avoiding these self-inflicted pains
  - single point of failure centralized network management also called "united we fall"
  - dead box deadlock
  - throttling slow host throttles fast host
  - missing message lost data
  - spoofing security breach
  - global autoloading load one—load all
  - the "bully" fixed trunk access priority
  - daisy chain NAD to trunk connection
  - resource deadlock no closure
  - error prone transmission requires higher level protocol
- d) and recognizing the inherent problems
  - error retries how many
  - resource allocation single path or multi-path
  - host/trunk (real/virtual channel) performance matching
  - the "missing ACK" resynchronization in the face of errors

Although the detailed analysis of these attributes is beyond the scope of this paper, they are shown to indicate considerations made in arriving at a working implementation.

#### LCN SITE PROTOCOL

Strictly speaking, Site Protocol is not one of the layers given in the ISO Open System Interconnection document. But it is presented to show that connectivity and accessibility are not the same thing. The configuration of trunk/

NAD interconnects, together with assignment of physical addresses and access codes, constitute the site protocol.

#### Connectivity

An example of LCN hardware is shown in Figure 2, where MF = mainframe and P = peripheral. The NAD can interface with one to four trunks, some examples of which are shown in Figure 1. The NAD connects its attached device to the network, but the interconnect pattern of the NADs/trunks defines the connectivity between devices. Note that attaching a device to the network does not imply connectivity with all other devices attached to the network. For example, MF/B cannot connect with P/C, P/D, or MF/E. Another example is MF/F, which can connect to all other devices except P/C.

#### Addressing

The hardware which interfaces a NAD to a trunk is called a Trunk Control Unit (TCU).

Each TCU is identified by an 8-bit physical address. The destination field of a message must match the physical address in order for the message to be accepted by the NAD.

The site protocol requires that all TCUs on a given NAD must have the same physical address.

#### Set identifier

Each NAD/trunk interface includes a 16-bit set identifier (the access code). The access code field of a message must match the switch selectable access code of the TCU in order for the message to be accepted by the NAD. The access code allows sets of NADs to share the same trunk yet be independent. All NADs of a set may access all other NADs within the set, but no others. Messages directed to a NAD outside of the set are not accepted by that NAD. Note that a NAD may be a member of more than one set. Access code transmission and matching are hardware functions.

#### Accessibility

These three characteristics (network connection, physical address, and set identifier or access code) constitute the accessibility of a NAD (and hence its attached host) to all other network NADs.

#### TRANSPORT CONTROL PROTOCOL

The Transport Control Protocol provides the method by which mutually accessible hosts exchange information. This protocol is defined independently of, yet with consideration for, the broad range of hosts considered likely LCN candidates.

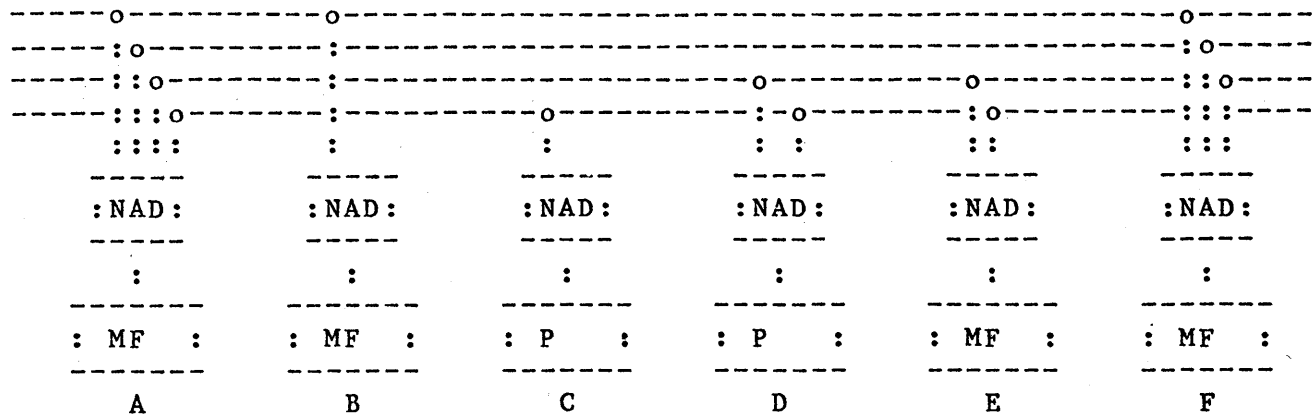


Figure 2—LCN connectivity.

### Data path

The primary means for information exchange is the data path. A data path is a logical, bi-directional “channel” terminating at each end in a host. Data path creation and deletion is performed by the NAD as directed by its host. A path normally is created between two different hosts, but the protocol allows a host to create a path to itself. Furthermore, some NADs are designed to have multiple attached hosts. When generating a path between hosts on the same NAD, or when a host creates a path to itself, the path is contained solely within the associated NAD since that NAD is attached to both path-end hosts.

More than one data path may exist between a pair of NADs.

A NAD can support up to 128 paths. However, since each path requires a small dedicated area in memory for path control (not including data buffers), the maximum number of paths supported by a NAD varies with its hardware configuration.

Information is passed on a path in the form of data, marks, or as a code.

### Data path-connect

Path creation is initiated by a host. A successful connect requires acceptance by the local and remote NADs (path control resource allocation), and by the destination host. That means the three intelligent entities—NADs and destination host—all have the opportunity to deny the request. Put another way, the three must cooperate in order to complete the connect. Once connected, a path exists until explicitly disconnected by either of the hosts, or until an unrecoverable error, such as a broken trunk, occurs.

At connect time the host presents the routing parameters to its NAD, or receives them from its NAD, depending on path-end. Both hosts also are given a path ID by their respective NADs. Thence forward, the hosts refer to the path by ID, and the necessary routing is automatically performed by the NAD.

### Data path-data exchange

By definition, data transfers are bi-directional on a data path. Host transfers are “blocked” by the NAD before transmission on the trunk in order to (a) decouple the host channel rate from the trunk rate and thereby allow unused trunk time to be used by other NADs, (b) to provide a mechanism whereby the data buffer area of NAD memory can be allocated dynamically across several paths, (c) to provide concurrent bi-directional transfer, and (d) to segment a long transfer for lower probability of induced errors.

The amount of data transferred is unlimited, but the effective rate of transfer is limited by buffer availability (real and path threshold) in the associated NADs, as well as by trunk loading (the “virtual channel” characteristic). A special data transfer mode is provided in which the trunk is not released between data blocks. In this mode only the two NADs involved can use the trunk, all other NADs being locked out. In this mode trunk loading is removed as a factor, and the network appears as a dedicated point-to-point connection. Transfer rates are then limited by the host channel and trunk rates and overhead.

### Data path-mark

A mark is a special form of data, analogous to an end of record mark. A mark may be sent at any time. The data and interspersed marks are delivered in the same order as sent.

### Data path-code

In addition to data and marks, a very short (32 bit) message may also be transmitted down a path. This message (CODE) is different from data and marks in two ways. First, the data path CODE buffers at each end are permanently allocated and, hence, always available (unlike data buffers which are dynamically allocated). Second, CODE messages are not queued at the receiving end; rather each CODE message “overwrites” the previously received one.



*Data path-disconnect*

A path may be disconnected by either path end host. The NAD performing the disconnect first transmits all of its outstanding output data, marks and code; then sends a disconnect message to the other NAD. Residual input is discarded in the NAD initiating the disconnect.

The disconnect is queued at the receiving NAD and is presented to the attached host after all data (and marks) in front of it have been presented.

*Control message*

The control message (DATAGRAM) is an alternate means for information exchange. Control Messages are self-contained, meaning they have fixed length, they contain routing information, and they are not associated with data paths. The host supplies the Control Message to the NAD, and the NAD simply sends it to the destination NAD. A Control Message, which for any reason cannot be delivered to the destination NAD, is returned to the host.

The control message format is shown in Figure 3.

SUMMARY

Transport Control Protocol uses data paths and Control Messages for information exchange between mutually accessible hosts. Since data paths exist as logical "channels," a host and its attached NAD may have many paths assigned and serviceable concurrently (but not simultaneously). At the same time the Control Message, independent of data paths, is available for flow control, status, functional requests of a higher level, or just general chit-chat.

NETWORK CONTROL PROTOCOL

This protocol level defines information flow control between NADs.

In order for a host with a dedicated NAD to exchange

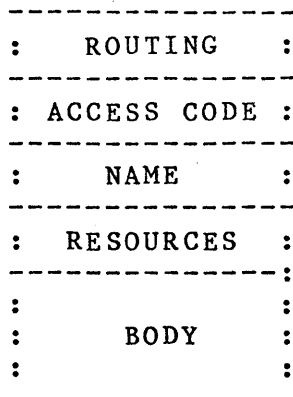


Figure 3—Control message.

information with any other host, the transfer obviously requires traversing a trunk from NAD to NAD. Hosts sharing a NAD, however, communicate with each other through their shared NAD, but with other hosts across a trunk (and another NAD).

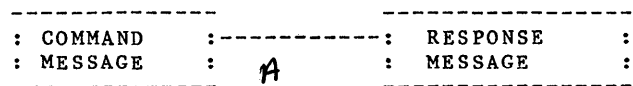
A primary task for the Network Control Protocol is recognizing and processing these routing variations.

*Command/response message modes*

Three modes are defined for communications between NADs. Each mode is tailored to a specific work function, to effectively use the trunk and NAD resources.

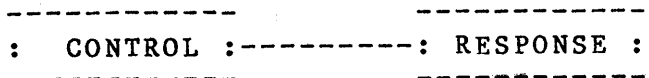
In the following three sections, the graphic conventions listed below are used.

The Command Message and Response Message, which always occur in pairs, are illustrated as a pair of boxes connected by a line.



The left hand box is always a command message, and the right hand box is always a response message. The interconnecting line represents the interval during which the receiving NAD processor examines the command message and determines the appropriate response message to return. The trunk remains captured until the processor signals the trunk interface to transmit the response message.

*Mode 1—control*



Mode 1 consists of one command/response message pair. CONTROL is directed to the host, and has meaning defined by a higher level protocol, or is a flow control message directed to and processed by the NAD processor.

*Mode 1 command messages*

- \*\* Control Message
- \*\* Connect
- Pathcode
- Path Mark
- Disconnect
- Double Purge
- Connect Accept
- Connect Reject
- Status Change

\*\* There are the only messages (of all three modes) which are generated by the host.

*Mode 1 response messages*

ACK  
 WAITNAK  
 Sequence Error  
 Illegal Command Message  
 Illegal Path

*Mode 2—data transfer*

```

-----
:  PATH  :-----:  ACK  :      :  PATH  :-----:  ACK  :
:  REQUEST :      :      :      :  DATA :      :      :
-----

```

Mode 2 is a two command/response message set used for transmission of data. The first command, Path Request, identifies the path and the buffer size. The receiving NAD returns an ACK if the data transfer is permissible. The sending NAD then immediately transmits the data, a closing ACK is returned, and the trunk is released.

If the receiving NAD cannot accept the data, it will return a NAK, following which the trunk is released.

*Mode 2 command messages*

Path Request  
 Path Data

*Mode 2 response messages*

ACK  
 Queue Full NAK  
 WAITNAK  
 Illegal Command Message  
 Illegal Path  
 Sequence Error

*Mode 3—captured trunk (streaming mode) data transfer*

```

-----
:  PATH  :-----:  ACK  :      :  DATA :-----:  ACK  :  o o  :  DATA :-----:  ACK  :-----:  DISABLE :-----:  ACK  :
:  REQUEST :      :      :      :  DATA :      :      :      :  STREAM :      :      :
-----

```

Mode 3 is a special form of data transfer in which trunk multiplexing is temporarily halted by capturing the trunk (streaming) for the duration of a multi datablock transfer. Consequently, total trunk bandwidth is allocated to the path capturing the trunk. Two useful effects result. First, the data path transfer rate is maximized since trunk loading has been eliminated and protocol overhead minimized. Second, all other NADs (and their hosts) on the trunk have had their intercommunication momentarily suspended, which implies an interlock capability.

*Mode 3 command messages*

Enable Stream Path Request  
 Disable Stream  
 Path Data  
 Path Mark  
 Path Code  
 Wait

*Mode 3 response messages*

ACK  
 BLOCK SEQUENCE ERROR  
 Illegal command message  
 Illegal Path  
 Disconnect  
 Ready  
 Nak

*Message transmission retry*

As explained in the Link Control Protocol section, all transmissions consist of a command and response message pair. A normal transmission consists of a TCU receiving a command message addressed to it, and returning a response to the TCU originating the command.

*Transmission abnormality*

A transmission abnormality occurs when no response is forthcoming to the command. Listed below are the main reasons for no response occurring:

1. Addressed a nonexistent NAD (actually TCU)
2. Command message garbled on the trunk causing
  - a. Destination Field to address a nonexistent NAD
  - b. Check sum error at receiver (no answer if check sum error)
3. Command message received correctly but the response message was garbled giving a similar effect as (2).

When a transmission abnormality occurs, the command message is retransmitted (with the same sequence number), up to 256 times, until a response message is received. If no response message is received after 256 retries, a fatal error has occurred. The disposition of the unsuccessfully transmitted command message depends on its type.

During retry the controlware will not attempt to send any command messages other than the one enduring the transmission abnormality. The controlware, however, will accept incoming command messages.

*Destination busy*

The status of the responding NAD (TCU) is included in a response message. One of the status bits is the "Memory

busy" bit. The "Memory busy" status signals that, although the command message was received by the destination TCU correctly, it could not be passed on to NAD memory. Hence, the response returned is likewise not from memory but generated by the TCU. Since the NAD processor at the destination did not receive the command, it must be retransmitted.

Provided the remainder of the response status is correct, the command message is queued for retry, and will be retransmitted until accepted by the destination NAD memory (or until a fatal error occurs).

The retry queue is serviced on a fixed interval basis. Other command messages may be sent during this interval.

#### *Destination fatal error*

Other NAD status bits in a response message include NAD processor not running, sequence errors, and other indicators of fatal errors at the destination NAD.

If the response status indicates a fatal error, the command message is not retransmitted. Marked "fatal error," the message is disposed of according to type.

#### *Fatal message errors*

Fatal message errors are those which are caused by hardware errors during the transmission of a trunk command/response. These errors can be caused by:

- trunk interface failures at either NAD
- trunk/data set failures
- NAD failures.

### LINK CONTROL PROTOCOL

The LCN Link Control Protocol defines a format and sequence of bits impressed upon the trunk to facilitate the transmission of information. The vehicle for all command and response information on the trunk is called a message. Each transmission on the trunk consists of only one message frame. In all cases, communication between two elements *X* and *Y* consists of a pair of message transmissions: a command message transmitted from *X* to *Y* and a response message transmitted from *Y* to *X*.

#### *Command message frame structure*

A valid command message is a minimum of ten 8-bit bytes in length following the frame synchronization sequence and

must conform to the following structures: *P, F, T, FUN, A1, A2, RP, S, L1, L2, FC1, FC2, I, FC3, FC4* where,

- P* = preamble of all ones preceding sync frame
- F* = message frame synchronizing byte
- T* = destination address byte
- FUN* = function byte
- A1, A2* = access code bytes
- RP* = resync parameter byte
- S* = source address byte
- L1, L2* = length field bytes
- FC1, FC2* = header frame check sequence bytes
- I* = information field, variable length
- FC3, FC4* = information frame check sequence bytes

Frames containing only link control sequences form a special case where no *I* field is present.

#### *Response message frame structure*

A valid response message is a minimum of ten 8-bit bytes in length following the frame synchronization sequence and must conform to the following structure: *P, F, T, FUN, P1, P2, P3, S, L1, L2, FC1, FC2, I, FC3, FC4* where,

- P1* = not used
- P2* = TCU/TCI status byte
- P3* = not used

and all other elements are identical to the command message elements.

### PHYSICAL LEVEL PROTOCOL

The transmission scheme employs carrier modulation of self clocked data, with the NADs attaching to a coaxial trunk via a T-tap. The measured error rate of the combination (including data sets) is  $10^{-12}$  or better within the configuration limitations.

Access to the trunk is governed by Trunk Reservation and Contention Elimination (TRACE) priority hardware. TRACE is a rotating priority scheme in which every NAD is given access to the trunk in turn.

### REFERENCES

1. Enslow, Jr., Philip H., "What is a Distributed Data Processing System?" *Computer*, January 1978, pp. 13-21.
2. ISO—International Organization for Standardization, "Reference Model of Open Systems Interconnection," ISO/TC97/SC16 N, August, 1979.

# LCN—A loosely coupled network system

by LOWELL H. SCHIEBE

Control Data Corporation  
Arden Hills, Minnesota

## INTRODUCTION

The Control Data Loosely Coupled Network (LCN) system is a set of hardware and software that interconnects computers and peripheral devices and provides them a means to communicate with each other over shared high-speed data trunks. The term Loosely Coupled is used to denote that there is no master/slave relationship in the LCN system. To effect communications between two units connected to LCN both units must agree to the transaction. This type of loosely coupled connection has been described previously<sup>1</sup>.

The conventional method of interconnecting computers with a high-speed channel is via point-to-point dedicated links where a connection consists of two adapters and a high-speed link as shown in Figure 1. This type of interconnection method is only reasonable if the number of computers is kept very small. The number of adapters and links required to interconnect all computers increases dramatically as units are added. A system of five computers, as shown in Figure 2, requires 20 adapters. The relationship between the number of computers, links and adapters can be described by the following formula.

$$N = D(D-1)$$

Where  $N$  = number of adapters required

$D$  = number of computers to be interconnected.

Whereas the conventional method requires many adapters for a large system, the LCN provides a means to interconnect many computers or peripherals with high-speed serial data trunks using only one adapter per device. A typical LCN system is shown in Figure 3. The Network Access Device (NAD) is the unit that adapts the unique computer or peripheral channel to the high speed serial data trunk. In the rest of this paper the term Device will refer to a computer or peripheral equipment that is attached to a NAD.

## LCN SYSTEM ARCHITECTURE

The system architecture of the CDC Loosely Coupled Network is based on logical rather than physical interconnects. The following four attributes describe the key features of the LCN architecture.

### *Concurrent logical paths*

One of the LCN attributes is that it has concurrent logical path capabilities. The system software, which is called controlware, contained in each NAD is based on a bi-directional logical path concept. With this concept, devices request logical paths to other units and then send/receive data and system messages over these paths. The NAD controlware establishes and maintains these logical path connections performing the logical/physical path transformations. When data is to be sent between devices across a logical path, the two NADs on each end of the path cooperate in obtaining use of the LCN trunk and in moving the data. The paths are bi-directional meaning that data can be sent in either direction between devices. Multiple logical path connections are provided in each NAD and data transfers can be active on several logical paths within each NAD.

### *Device independence*

A second attribute of the LCN is that it is independent of the devices and/or systems attached to it. The controlware in each NAD allocates and controls the NAD resources. The NAD controlware also establishes the physical routings between devices that correspond to the system logical path connections and moves data across these path connections. The LCN/NAD is independent of the data being transferred across these paths and does not examine or use the data being moved. Buffering within the NAD decouples the device transfer rates from the LCN data transfer rate.

### *Connectivity*

A third attribute of the LCN is that it interconnects many devices via a set of common high-speed serial trunks and NADs. Each high-speed serial trunk can interconnect 32 units and the NAD can be connected to four of these trunks. Using all of the possible connections, a device can be interconnected to 124 other devices using only one NAD and four trunks.

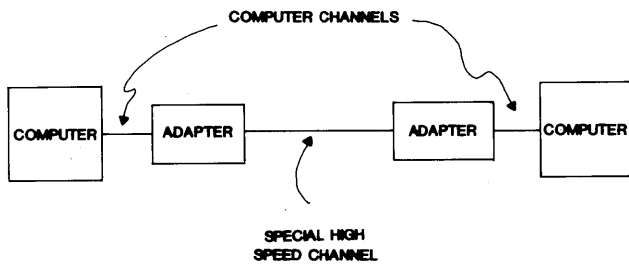


Figure 1—Point-to-point connection—two computer system.

**Reliability**

The fourth attribute of the LCN is that it has no single point of control making it inherently more reliable. Control of the LCN is distributed among all NADs such that each controls only its own resources. Failure of a single NAD thus has minimal affect on the LCN system. The connectivity provided by LCN permits redundant systems to be easily configured. In each NAD the controlware performs integrity checks and handles error recovery. Errors detected on the trunk are automatically retried by the NAD before the device is notified. Status tables and error logs are kept in each NAD for tracking LCN performance.

**LCN SERIAL TRUNK HARDWARE**

The serial trunk transmission system consists of a 50 megabit data set, power-splitting T-Tap and coaxial cable. The data set uses a phase modulated carrier system to transmit data in a synchronous burst mode. High quality coax cable and connectors are used to minimize possible ground and EMI/RFI problems. The power-splitting T-Tap is a passive device that connects the data set to the trunk and provides substantial signal isolation between the trunk and data set. The isolation allows units to be powered down and removed from the system without affecting the integrity of the main trunk or disrupting the entire LCN system. The data set, coax cable and T-Tap system provide a 50 megabit/sec transmission rate and allow up to 32 data set attachments per

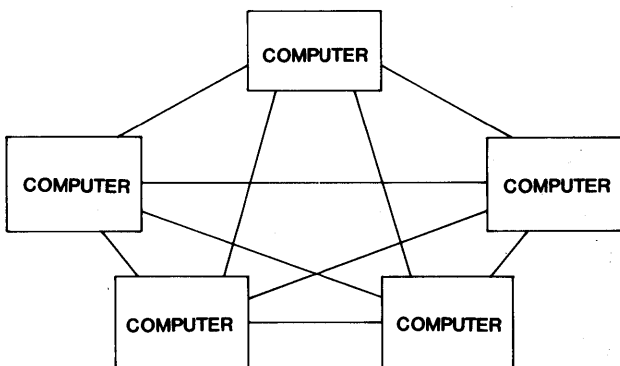


Figure 2—Point-to-point connection—five computer system.

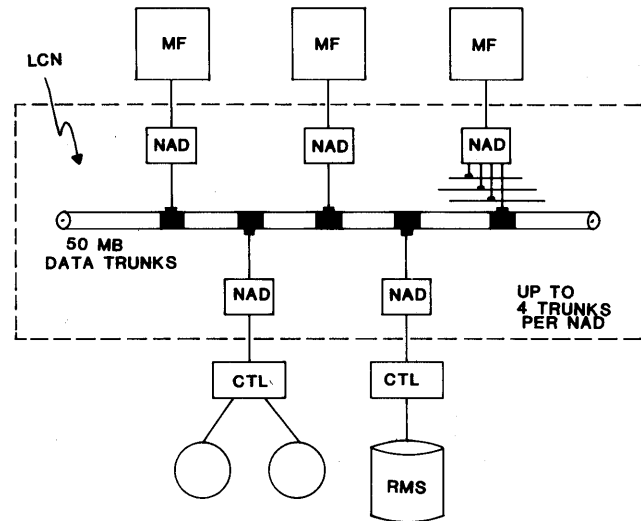


Figure 3—Typical LCN system.

serial trunk. A maximum trunk length of 1000 feet is allowed with 16 attachments; however, longer lengths are possible with fewer attachments.

**NETWORK ACCESS DEVICE**

The Network Access Device (NAD) consists of four functional elements. Three elements are common to all NADs and one element is unique to the device or channel being interfaced. A block diagram of the NAD is shown in Figure 4.

The NAD internal bus is used for inter-element communication. Bus usage is allocated equally among three elements: The trunk interface, the processor, and the device interface. Time-slice allocation allows each of these three elements to access the bus, and therefore the memory, at a guaranteed 50 Mbps rate (16 data bits every 320 nano-seconds).

*NAD trunk interface*

The trunk interface element consists of hardware and micro code that matches the NAD both electrically and logically to the high-speed (50 megabits per second) serial trunk. The trunk interface function is divided into two pieces; the Trunk Control Unit (TCU) and Trunk Control Interface (TCI).

The TCU interfaces the data set to the TCI and adds/deletes the serial trunk protocol envelope which includes Cyclic Redundancy Code (CRC) generation and detection. The TCU also interprets serial trunk message functions and reacts accordingly, generating response messages to ensure closure for all valid incoming messages. Contention for serial trunk access is also provided in the TCU.

Contention for trunk access is resolved by a rotating priority mechanism<sup>2</sup>. This mechanism prevents contention and

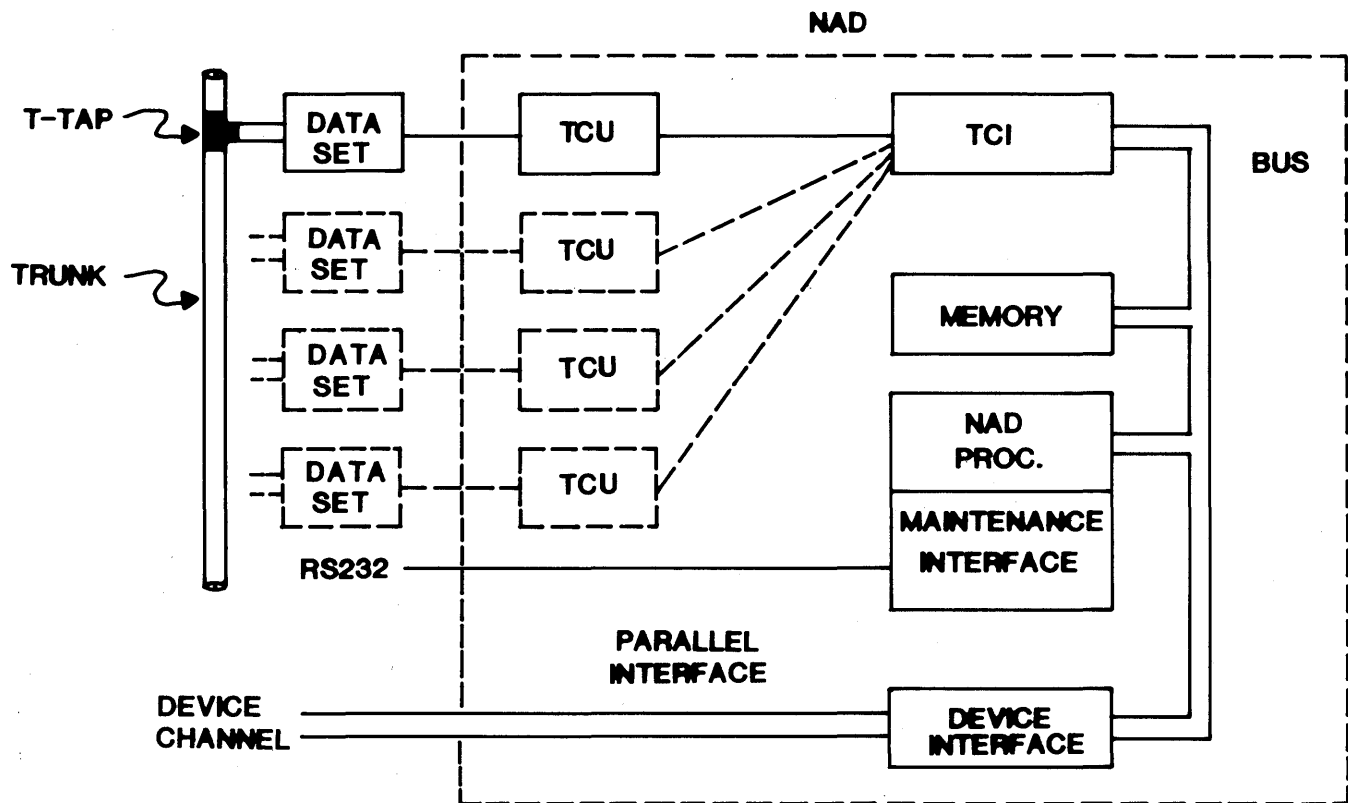


Figure 4—NAD block diagram.

guarantees trunk access to all units on the trunk even during peak loading of message mode traffic by rotating the access priority between all NADs on a trunk. A special message transfer mode called streaming is provided for cases when a high transfer rate is required.

The TCI interfaces up to four TCUs depending on the number of serial trunks that are connected to the NAD. The function of the TCI is to control data/message transfers between the active TCU and NAD memory via the NAD internal bus. The TCI also resolves all transmit request/receive message and multiple TCU request conflicts. TCI operations are directed by the NAD processor via commands stored in NAD memory.

#### *NAD memory*

The NAD memory provides for storage of controlware programs and data buffers. It also acts to buffer data-rate differences between the synchronous serial trunk and asynchronous device. The size of the memory depends upon the attached device and particular system configuration. The maximum memory configuration is 128K 8-bit bytes.

#### *NAD processor*

The NAD processor is a 16 bit wide interrupt driven processor. It is constructed of four bit microprocessor chips

which are microcoded to yield a macro instruction set. The NAD processor executes controlware residing in the NAD memory to provide management of the NAD resources, management of data flow through the NAD which includes initiation of I/O transfers on both TCU/TCI and device interfaces, and execution of LCN system functions. The serial RS232C maintenance interface provides a maintenance connection into the NAD.

#### *NAD device interface*

The NAD device interface is unique for the particular device or channel being interfaced. The device interface adapts the device channel electrical signals to the NAD and provides data assembly/disassembly to handle different word sizes. Device interface operations such as transferring data and commands between an attached device and the NAD memory are directed by the NAD processor via commands stored in NAD memory. If the attached device is passive such as a disk, tape, etc., device control is also provided. Up to four device channel connections are available on some device interfaces.

#### NAD CONTROLWARE

NAD controlware is the set of software that resides in the NAD memory and is executed by the NAD processor. This

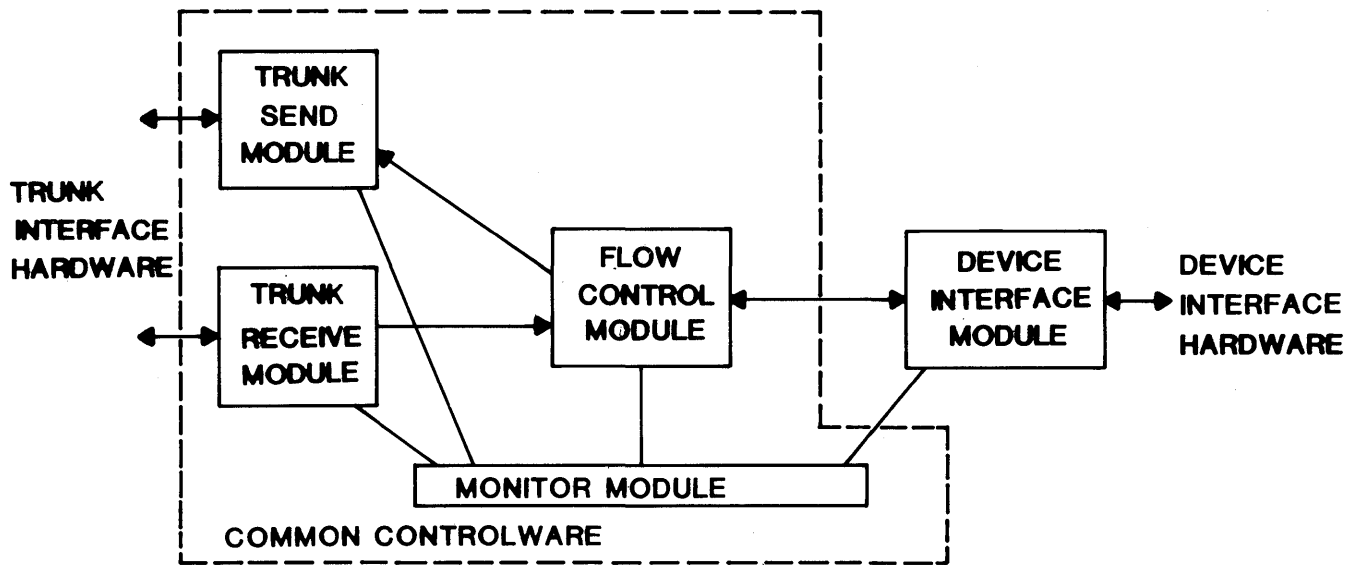


Figure 5—Controlware block diagram.

controlware implements the four lower level protocols, (Transport, Network, Data Link, and Physical) defined in the ISO Open Systems Interconnection Model.<sup>3</sup> The higher three levels (Application, Presentation and Session) are implemented within the attached host computer.

NAD controlware consists of a set of common controlware modules and a unique device interface module. A block diagram of the controlware is shown in Figure 5.

#### *Common controlware*

The common controlware consists of four modules that provide control of the NAD including management of internal NAD resources, control of data flow between the device interface and serial trunk interface, initiation of recovery procedures for errors, and gathering of statistics. The Trunk Send module controls the sending of messages and receiving of responses on the trunk interface which includes setting up the hardware interface control. The Trunk Receive module controls the receiving of messages and sending of responses on the trunk interface including setup of the hardware interface control. Flow Control is the module that interfaces the Trunk Send and Trunk Receive modules to the Device Interface module controlling all data movement through the NAD. The Monitor module controls linkage of controlware modules, handles NAD interrupt state changes and contains general utility and initialization programs.

#### *Device interface controlware*

The unique NAD controlware module is the Device Interface. This module controls data flow between the device

channel and the NAD memory. Besides the interface control, this module also handles the higher level protocol interface. If the device is a computer, the controlware interfaces to the computer's system software. If the device is an I/O unit or controller, the Device Interface module also performs the "typical" I/O driver function. This I/O driver function involves interpreting the higher level protocol requests for I/O activity and converting them to unique sets of function codes or commands for the I/O device. The I/O driver function also includes error recovery on the I/O device.

#### SUMMARY

The Control Data Loosely Coupled Network system provides a mechanism to interconnect many devices (computers and peripherals) using Network Adapter Devices and high speed (50 megabit per second) serial trunks. The serial trunk system provides connectability between many devices and the NAD hardware provides independence from the attached device characteristics. Control of the LCN system is distributed among the NADs, providing independence from a single point of control (failure). The logical path scheme provides additional independence from the attached device software. Concurrent data transfers on multiple logical paths provide enhanced performance capabilities.

#### REFERENCES

1. Enslow Jr., Philip H., "What is a Distributed Data Processing System?" *Computer*, January 1978, pp. 13-21.
2. Burke, R. G., "Eliminating Conflicts on a Contention Channel," 4th Conference on Local Computer Networks, October 1979, pp. 48-55.
3. ISO—International Organization for Standardization, "Reference Model of Open Systems Interconnection," ISO/TC97/SC16 N, August 1979.

# Derivation and use of a survivability criterion for DDP systems\*

by RICHARD E. MERWIN and MOHAMMED MIRHAKAK

*The George Washington University  
Washington D.C.*

## INTRODUCTION

With the advent of low cost compact computing systems, there has been a natural tendency to bring the computer to the job rather than the job to the computer. The geographical distribution of computing also leads to a necessity to interconnect installations so that they can share data, provide back up computing support, and permit rapid transfer of messages between sites. We are now seeing the emergence (1,2) of networks of computers interconnected by communication facilities. There are numerous benefits to be achieved by this dispersal of computing facilities including more reliable overall operational capability, better overall service to geographically separated sites, and the advantages of being able to utilize a wide range of software and hardware facilities available through telecommunication systems.

Inherent in a distributed data processing (DDP) system is to a varying degree some interdependence of each computing site and the interconnecting communications system. Failure of a particular DDP network computer site, hereafter referred to as a node, will have a negative effect on the overall distributed data processing system. In a similar vein, failure of communication links will reduce the performance of the system. A study was initiated to examine these failure modes and develop criteria to measure the performance of a DDP system, including its associated data distributions, in terms of individual equipment failure modes and associated probabilities. The term survivability index is used as a performance parameter of a DDP system and an objective function has been defined to provide a measure of survivability in terms of the nodes and links of a network and their failure probabilities, data set distributions, and weighting factors for network nodes and computer programs.

Having derived an objective function to measure DDP performance, alternative data set distributions and network architectures can be evaluated. Criteria can be included such as addition or deletion of communication links, movement of programs among nodes, duplication of data sets, etc. Constraints can be introduced which limit the number and size

of files and programs that can be assigned to a node and the maximization of the objective function will be subject to these constraints. One of the key concerns in our derivation of an objective function is its computability. The present algorithm exhibits exponential growth with the number of nodes and links. Studies are in process to find more efficient computational algorithms to quantify the survivability index.

This paper is organized into eight sections. The next three sections will provide background on communication network survivability, DDP survivability concepts, and a definition of a survivability index  $S$ . An objective function is derived to quantify the DDP survivability index  $S$  in the fifth section, followed by a section presenting an example computation of  $S$  for a four node network. The next section describes the evaluation of  $S$  using a computer program for nine node networks representing seven architectures with three data distributions. Some comments and conclusions are presented in the final section.

## BACKGROUND: COMMUNICATION NETWORK ANALYSIS

The study of communication network survivability (3,4,5) has been divided into two nearly disjoint areas: deterministic and probabilistic. This activity was aimed at determining optimal communication network architectures to better withstand either wartime attack or the impact of natural disasters. Since it serves as a starting point for the research activity described below for a DDP system, a brief review of the underlying concepts of these studies is presented here.

Deterministic survivability (3,6,7,8,9) presumes a fixed communication network architecture. In considering survivability from wartime attack, it is assumed the adversary has full knowledge of the network architecture. Survivability is measured in terms of maximal connected subnetwork components surviving after damage is inflicted on the original communications network. Criteria include the number of nodes still in communication after removal of nodes and links. Selection of network architectures which maximize the connected subnetwork components are the goal of these studies (3,4,5).

The approach to defining probabilistic survivability of a

\* The Research described here was partially supported by the Defense Civil Preparedness Agency, Washington, D.C. under Contract DCPA01-78-C-0271.



communication network can be further divided into networks exhibiting random structure (3,10,11,12,13) or fixed structure (3,14,15,16,17). For networks with random structure it is assumed that any two nodes are connected with a known probability. This probability can be a function of whether the network is subjected to attack or a natural disaster. Survivability is measured in terms of the probability of survival of a connected network, i.e., one in which some specified number of nodes are still in contact. Mathematical techniques of cut sets and path analysis are used to determine survivability of a damaged network.

The fixed structure approach assumes a fixed and known network architecture and assigns a probability of failure to each node and link. Again these probabilities can be raised or lowered as a function of being subject to wartime attack or natural disasters. Survivability is measured in terms of the probability of criteria such as "all nodes can communicate with one another" or that a "percentage of nodes both survive the attack and remain in contact with the largest single group (component) of surviving nodes" (3).

#### DISTRIBUTED DATA PROCESSING SYSTEMS

DDP systems can be constructed in a number of ways. A very simple example occurs when a data processor serves as a "front end" for a larger processor and handles input-output functions. A somewhat more complex example is an array of interconnected processors (18,19). More complexity is added when geographically remote processors are interconnected by a telecommunications network (1). It will be this concept of a DDP which is considered in this paper.

Adding data processing and associated data sets at the nodes of a telecommunication network to create a DDP system greatly complicates the analysis of survivability. If it is further assumed that not all data sets are co-resident with the processor executing a program at a node, the quantification of DDP survivability becomes even more difficult. It is this latter implementation of a DDP system which is the basis for the research reported here.

When an executing program at a node needs access to another node within the DDP, two failure modes must be considered. One is that at least one link or node along all paths interconnecting the needed data to the executing program have failed thus breaking communication between the executing program and its required data. The second is that the node at which the required data is resident has failed. Either of these failure modes prevents a program at some node in the DDP from executing. We will refer to this situation as a DDP system with remote data requirements.

A final failure mode is that the node at which a program is to execute fails, i.e., the data processor is inoperative. Any of the three failure mechanisms noted above can cause a program at a node to be unexecutable. Probabilities for failure "q" and being operational "p" can be assigned to each node and telecommunication link. It is assumed here that these probabilities are independent which is consistent with the assumptions for survivability analysis for telecommunications networks.

#### DDP SURVIVABILITY CRITERIA

A simple quantitative measure of the survivability of a DDP is the number of programs that remain operational after some combination of nodes or links have failed. For a given network architecture there are a large number of subarchitectures that occur because of failures of nodes and links. Each of these subarchitectures has a probability of occurring and for each the number of operational programs can be determined based upon the data set requirements for programs executing at operable nodes and the data set distribution across nodes. A survivability criterion can be generated by taking the expectation of the number of programs operable for each subarchitecture leading to a summation of the proportion of programs operable for a subarchitecture times the probability of its occurrence. This criterion is designated as  $S$  and provides a quantitative measure of DDP survivability as a function of initial network architecture, a given data set distribution, and the data set requirements for each program at each node. A brief mathematical derivation of  $S$  is presented in the next section. An example of the use of this criterion for a simple four node network is described in the following section.

The mathematical approach to quantifying DDP survivability is computationally costly. For a DDP with  $N$  nodes and  $L$  links,  $2^{N+L}$  possibilities exist for subarchitectures. The requirement to access data at remote nodes greatly reduces the number of cases that must be considered but computationally we are quite limited as to the size of a DDP system that can be analyzed. At present we can handle DDP systems with nine nodes and from 10 to 12 links. Better mathematical approaches are being developed, but haven't been implemented in a computer program which calculates  $S$  for a given DDP and data distribution.

Other factors that can be introduced, which will impact  $S$ , are the assignment of weights to programs and nodes, and the introduction of constraints on the data set distribution. Weighting factors are introduced to indicate the relative importance to the function of a DDP system of individual programs or nodes. These weights would have to be assigned by the DDP designers and provision is made for incorporating these weights in the computer program which computes  $S$ .

The data set distribution constraints would again be design decisions and represent situations where only so many data sets could be resident at a node, or due to update frequencies, it would not be feasible to make duplicate copies of a data set which is generated at a particular node. In the DDP survivability analysis for various network architectures described below, both program and node weighting factors are held constant and equal while data set constraints are not considered.

#### SURVIVABILITY INDEX COMPUTATION ALGORITHM

An enumerative technique has been selected to evaluate the survivability index  $S$  for a given DDP system architecture

and data set distribution. The algorithm executes in four phases. The first phase determines the connected components, i.e., subarchitectures of the DDP network. The computation in this step is exponential in determining execution time and constitutes the limiting factor on DDP networks that can be analyzed for survivability. The second phase introduces the data set distribution and a further selection is made of those subarchitectures for which programs can execute, i.e., have access to the required data sets. A node is considered as surviving if at least one program executes at this node. The third phase assigns weighting factors to programs and nodes, while in the last phase the survivability index is computed by a procedure to be described below.

Given a DDP network consisting of  $N$  nodes and  $L$  links, define stochastic variables  $E_{ij}$  for every link  $(i,j)$  and  $E_i$  for every node  $i$ . These variables assume the values  $\{0,1\}$ . The value one indicates that the component is operational, while the value zero indicates it is nonoperational. The distribution of  $E_{ij}$  and  $E_i$  is:

$$P(E_{ij}=1) = p_{ij}$$

$$P(E_{ij}=0) = 1 - p_{ij} = q_{ij}$$

and

$$P(E_i=1) = p_i$$

$$P(E_i=0) = 1 - p_i = q_i$$

Assume that a DDP network consists of  $N$  nodes and  $L$  links. With  $N+L$  binary stochastic variables,  $2^{N+L}$  elementary events have to be considered. Each event corresponds to a subgraph, i.e., subarchitecture, of the graph corresponding to the network architecture. Each event's probability, since it is assumed all variables are independent, is the product of probabilities of the corresponding node and link probabilities. For event  $j$ , assume a combination of programs survives at each node. Let  $a_i(j)$  represent the combination of programs surviving at node  $i$  for event  $j$  and let  $b(j)$  represent the combination of nodes that survive for this event. Because of the dependence of survivability on the combination of programs and nodes surviving, the survivability is defined for event  $j$  and then expanded to find the total survivability for the whole network. For event  $j$ , the survivability index would be the weighted sum of the survivability of nodes as follows:

$$S^j = \sum_{i=1}^N W_i^{b(j)} S_i^j \quad (1)$$

where  $S_i^j$  is the survivability of node  $i$  for event  $j$  and  $W_i^{b(j)}$  is the weight of node  $i$  for combinations of surviving nodes  $b$  that depends on event (subgraph) number  $j$ .  $S_i^j$ , the survivability index of node  $i$  for event  $j$ , is the sum of the weights of the surviving programs at node  $i$ , as follows:

$$S_i^j = \sum_{k=1}^{m_i} W_{i,k}^{a_i(j)} \quad (2)$$

where  $m_i$  represents the number of programs at node  $i$ ,  $W_{i,k}^{a_i(j)}$  is the weight of program  $k$  located at node  $i$  for com-

bination of surviving programs  $a_i$  that depends on event number  $j$ . By substituting for  $S_i^j$  from (2) into (1) we would have:

$$S^j = \sum_{i=1}^N W_i^{b(j)} \sum_{k=1}^{m_i} W_{i,k}^{a_i(j)} \quad (3)$$

The total survivability is by definition

$$S = \sum_{j=1}^{2^{N+L}} P(A_j) \cdot S^j$$

where  $A_j$  represents the event  $j$  and  $P(A_j)$  represents the probability that  $A_j$  happens. The example showing the calculation of  $S$  in the next section will further clarify the details of the algorithm for computing the DDP system survivability.

### COMPUTATION OF $S$

An example of the computation of survivability index  $S$  for a simple DDP system is presented in this section. The DDP network shown in Figure 1 consists of 4 nodes and 4 links. Assignment of files and programs to nodes is as shown. FA denotes the files available while FN denotes the files needed to execute the programs at a node. PM designates the programs to be executed at a node. For example, at node 2 we have two programs,  $X_{2,1}$  and  $X_{2,2}$ . Files 2 and 4 are

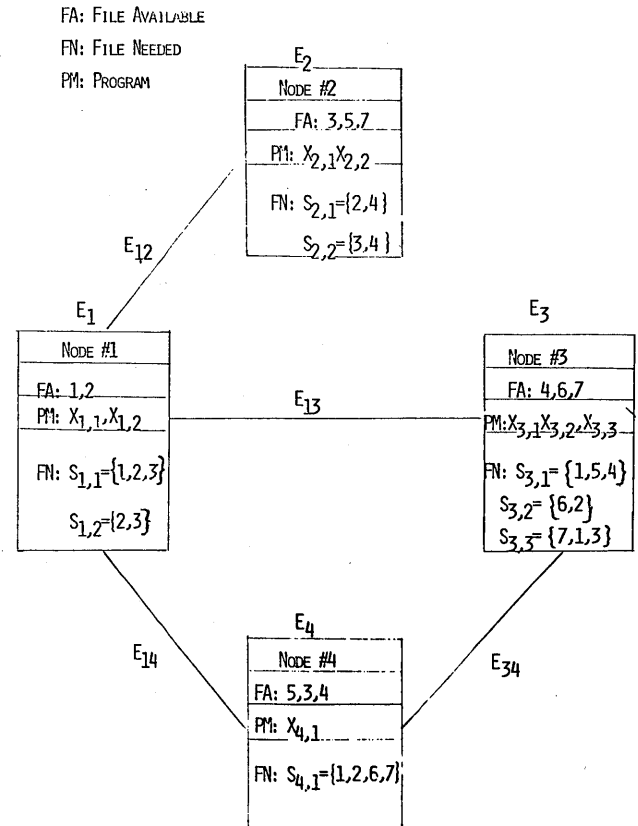


Figure 1—Four node DDP.

Item	E <sub>1</sub>	E <sub>2</sub>	E <sub>3</sub>	E <sub>4</sub>	E <sub>12</sub>	E <sub>13</sub>	E <sub>14</sub>	E <sub>34</sub>	X <sub>11</sub>	X <sub>12</sub>	X <sub>21</sub>	X <sub>22</sub>	X <sub>31</sub>	X <sub>32</sub>	X <sub>33</sub>	X <sub>41</sub>	Survivability and	comment
1	0	0	0	1	-	-	-	-	0	0	0	0	0	0	0	0	0	None of the programs survive
2	0	0	1	0	-	-	-	-	0	0	0	0	0	0	0	0	0	
3	0	0	1	1	-	-	-	1	0	0	0	0	0	0	0	0	0	
4	0	1	0	0	-	-	-	-	0	0	0	0	0	0	0	0	0	
5	0	1	0	1	-	-	-	-	0	0	0	0	0	0	0	0	0	
6	0	1	1	0	-	-	-	-	0	0	0	0	0	0	0	0	0	
7	0	1	1	1	-	-	-	1	0	0	0	0	0	0	0	0	0	
8	1	0	0	0	-	-	-	-	0	0	0	0	0	0	0	0	0	
9	1	0	0	1	-	-	1	-	1	1	0	0	0	0	0	0	0	1/4(1/2+1/2)(.9)3(.1) <sup>2</sup> X <sub>11</sub> & X <sub>12</sub> Survive
10	1	0	0	1	-	-	0	-	0	0	0	0	0	0	0	0	0	
11	1	0	1	0	-	1	-	-	0	0	0	0	0	0	0	0	0	
12	1	0	1	1	-	1	1	1	1	1	0	0	1	1	1	1	1	(1/4(1/2+1/2)+1/4(1/3+1/3+1/3)+1/4)(.9) <sup>6</sup> (.1)
13	1	0	1	1	-	1	1	0	1	1	0	0	1	1	1	1	1	(1/4+1/4+1/4)(.9) <sup>5</sup> (.1) <sup>2</sup>
14	1	0	1	1	-	1	0	1	1	1	0	0	1	1	1	1	1	3/4(.9)(.1) <sup>2</sup>
15	1	0	1	1	-	1	0	0	0	0	0	0	0	0	0	0	0	0
16	1	0	1	1	-	0	1	1	1	1	0	0	1	1	1	1	1	3/4(.9) <sup>5</sup> (.1) <sup>2</sup>
17	1	0	1	1	-	0	1	0	1	1	0	0	0	0	0	0	0	1/4(1/2+1/2)(.9) <sup>4</sup> (.1) <sup>3</sup>
18	1	0	1	1	-	0	0	1	0	0	0	0	0	0	0	0	0	0 Ignore the case E <sub>34</sub> = 0
19	1	1	0	0	1	-	-	-	1	1	0	0	0	0	0	0	0	1/4 (.9) <sup>3</sup> (.1) <sup>2</sup>
20	1	1	0	0	0	-	-	-	0	0	0	0	0	0	0	0	0	0
21	1	1	0	1	1	-	1	-	1	1	1	1	0	0	0	0	0	(1/4(1/2+1/2)+1/4(1/2+1/2))(.9) <sup>5</sup> (.1)
22	1	1	0	1	1	-	0	-	1	1	0	0	0	0	0	0	0	1/4(1/2+1/2)(.9) <sup>4</sup> (.1) <sup>2</sup>
23	1	1	0	1	0	-	1	-	1	1	0	0	0	0	0	0	0	1/4(.9) <sup>4</sup> (.1) <sup>2</sup>
24	1	1	0	1	0	-	0	-	0	0	0	0	0	0	0	0	0	0
25	1	1	1	0	1	1	-	-	1	1	1	1	1	1	1	0	0	(1/4+1/4+1/4)(.9) <sup>5</sup> (.1) <sup>1</sup>
26	1	1	1	0	1	0	-	-	1	1	0	0	0	0	0	0	0	1/4(.9) <sup>4</sup> (.1) <sup>2</sup>
27	1	1	1	0	0	1	-	-	0	0	0	0	0	0	0	0	0	0
28	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	(.9) <sup>8</sup>
29	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	(.9) <sup>7</sup> (.1) <sup>1</sup>
30	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	(.9) <sup>7</sup> (.1)
31	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	0	0	(3/4)(.9) <sup>6</sup> (.1) <sup>2</sup>
32	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	(.9) <sup>7</sup> (.1)
33	1	1	1	1	1	0	1	0	1	1	1	1	0	0	0	0	0	(1/4+1/4)(.9) <sup>6</sup> (.1) <sup>2</sup>
34	1	1	1	1	1	0	0	1	1	1	0	0	0	0	0	0	0	1/4(.9) <sup>6</sup> (.1) <sup>2</sup>
35	1	1	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	1/4(.9) <sup>5</sup> (.1) <sup>3</sup>
36	1	1	1	1	0	1	1	1	1	1	0	0	1	1	1	1	1	3/4(.9) <sup>7</sup> (.1) <sup>1</sup>
37	1	1	1	1	0	1	1	0	1	1	0	0	1	1	1	1	1	3/4(.9) <sup>6</sup> (.1) <sup>2</sup>
38	1	1	1	1	0	1	0	1	1	1	0	0	1	1	1	1	1	3/4(.9) <sup>6</sup> (.1) <sup>2</sup>
39	1	1	1	1	0	1	0	0	0	0	0	0	0	1	0	0	0	1/4(1/3)(.9) <sup>5</sup> (.1) <sup>3</sup>
40	1	1	1	1	0	0	1	1	1	1	0	0	1	1	1	1	1	3/4(.9) <sup>6</sup> (.1) <sup>2</sup>
41	1	1	1	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1/4(.9) <sup>5</sup> (.1) <sup>3</sup>
42	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0

S = .7658

There are 42 cases out of 256 (2<sup>8</sup>=256).

Figure 2—Example of computation of S.

required to execute program  $X_{2,1}$  and are designated as  $S_{2,1} = [2,4]$ .

If a program is executable at a node, then that node exhibits some degree of survival. In the DDP system shown in Figure 1, no program can execute at a node without access to a file resident at some other node. Thus for program  $X_{2,1}$  to execute at node 2, it must have access to file number 4 at node 3 or 4. This implies that data link  $E_{12}$  and node 1 be operational. In addition, either data link  $E_{13}$  and node 3 or data link  $E_{14}$  and node 4 must be operational. As can be seen, the same conditions exist for program  $X_{2,2}$  to be executable at node 2.

For the particular DDP shown in Figure 1 there are 256 different configurations ranging from everything having failed (all nodes and links) to everything being operational. For many of the possible configurations of operable nodes and links no program can execute at any node. The first phase of our computational algorithm selects all operable network subarchitectures for which it is possible for one or more programs to operate at a node. For Figure 1 there are 42 such configurations as shown in Figure 2 which illustrate the computational steps for  $S$ . When availability of data sets are included as a selection criteria, then only 26 of the 42 operable subarchitectures have nodes with an executable program. The probability of occurrence of each of the 26 subarchitectures times their weighting functions is shown on the right side of Figure 2. Survivability index  $S$  is the summation of these terms.

For purpose of illustration of this computation, the probability of survival for a node or link was set at the representative values at 0.9, and for failure at 0.1. All nodes and programs were assumed to be of equal weight. Since the sum of the weights of all nodes must equal one, the weight for each of the four nodes is  $\frac{1}{4}$ . Likewise, since the sum of the weights of the programs at each node must equal one, the program weights  $W_i$  are:  $\frac{1}{2}$  at node 1, i.e., 2 programs resident at this node;  $\frac{1}{2}$  at node 2;  $\frac{1}{3}$  at node 3; and 1 at node 4. Using the definition of survivability  $S$  as noted above, we get  $S = 0.7658$  for the DDP system shown in Figure 1.

#### DDP NETWORK SURVIVABILITY COMPARISON

A computer program has been generated which computes  $S$  for a given DDP network, and program and data set distribution. The use of this program is restricted to DDP systems where the sum  $N+L$  is less than 20. For DDP of this complexity the running time of this program on an IBM 370/148 is several minutes. The addition of a node or link increases the running time by factors approaching two. No attempt has been made at this time to improve the efficiency of the program to enable computation of  $S$  for larger DDP networks. As noted above, other mathematical algorithms to reduce computation time and significantly increase the size of a DDP system for which the survivability index can be computed are being developed as part of an ongoing research program.

Seven DDP network geometries were investigated as shown in Figure 3. Some familiar DDP architectures which

have been evaluated for survivability include the ring (3f), star (3e), grid (3a), and linear Z (3b). A single program is assumed resident at each node and a set of data files required to run each program is specified. Three distributions of data files were specified. Two were selected at random while the third was chosen to minimize the distance in terms of links between a program and the data files required to execute the program. Table I shows the three data-file node assignments and indicates for each node what data must be fetched from a remote node.

The corresponding value of survivability index  $S$  for each of the test cases (21 in all) were computed. Table II contains a tabulation of values of  $S$  for each test case.

As was expected, the data file assignment can have a large impact on DDP system survivability, thus demonstrating the importance of this factor in the design of distributed computer networks. Individual DDP architectures varied widely in terms of survivability when the data set distribution was changed. As would be expected, architectures with the most links, i.e., Figure 3a—grid, had better survivability. Also as was expected, the star architecture has high survivability because the distance between a program and its data set never exceeds two links. The various ring architectures rank moderately high on survivability. The string architectures, i.e., linear Z and swastika, have fewer links (8) and correspondingly lower survivability. This reflects the greater distance that can occur between data sets and programs and the lack of parallel or redundant communication paths between nodes.

#### CONCLUSIONS

The research directed toward deriving a survivability criteria in this paper for computer networks is still in a very preliminary stage. A survivability index  $S$  has been defined which measures survival in terms of the number of programs which remain executable in the DDP after some nodes or

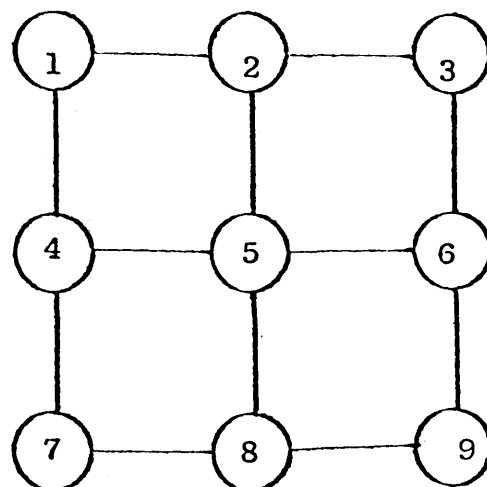


Figure 3a—Grid

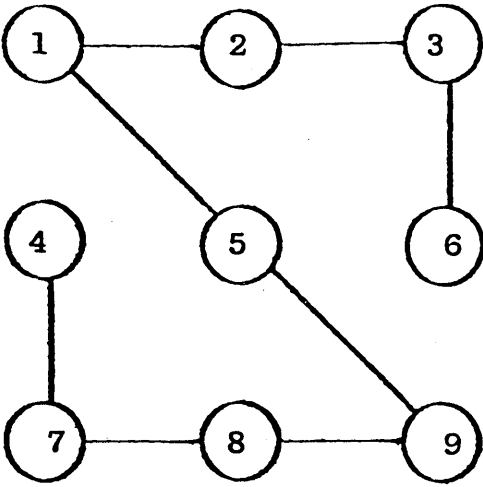


Figure 3b—Linear Z

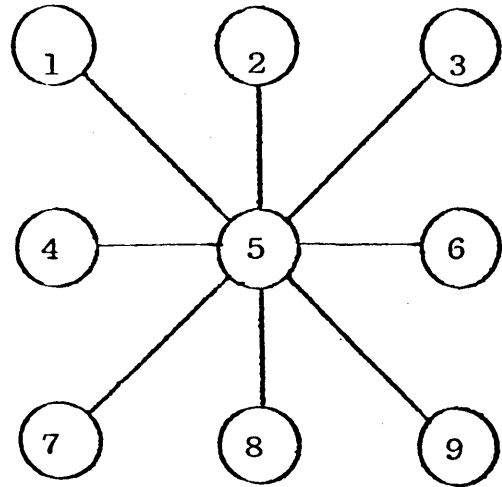


Figure 3e—Star

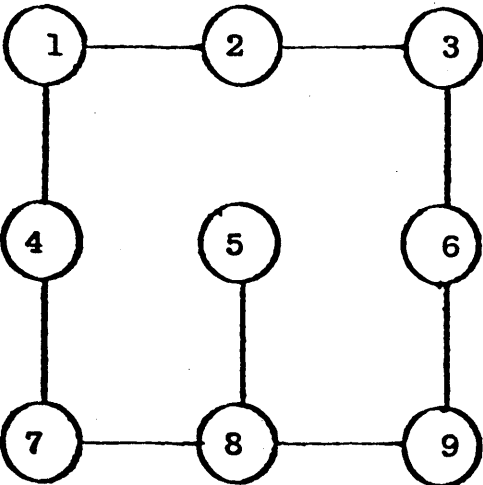


Figure 3c—Ring w/tail

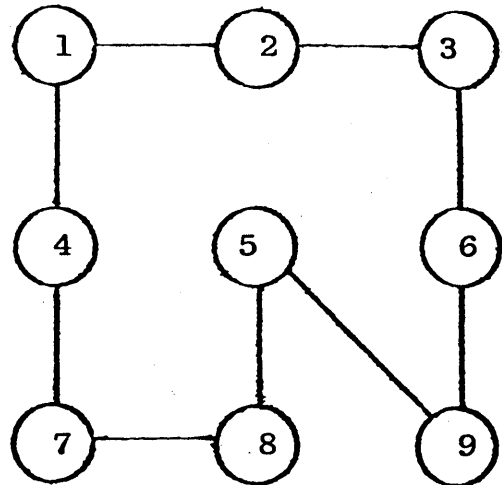


Figure 3f—Ring

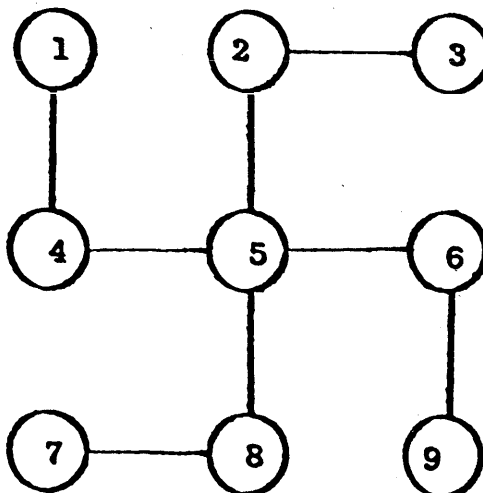


Figure 3d—Swastika

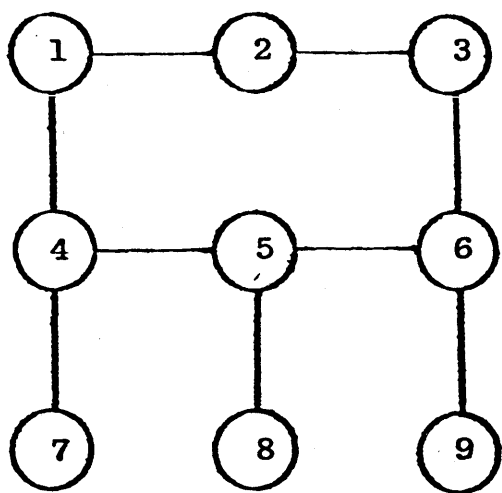


Figure 3g—Ring w/3 tails

TABLE I.—Data file distribution

Node Number	Resident Program Number	Files Needed	Distribution 1		Distribution 2		Distribution 3	
			Resident Files	Non-Resident Files	Resident Files	Non-Resident Files	Resident Files	Non-Resident Files
1	1	1,3,8	1	3,8	3	1,8	1,3	8
2	2	1,4,6	4	1,6	7	1,4,6	6	1,4
3	3	2,3,7	2,7	3	2,5	3,7	7	2,3
4	4	1,4,6	1,6	4	1,4,6	-	1	4,6
5	5	5,7,8	1,5	7,8	1,8	5,7	4,8	5,7
6	6	2,3,5	3	2,5	6,7	2,3,5	2,5	3
7	7	4,5,8	5,8	4	5	4,8	5	4,8
8	8	2,3,6	2,6	3	1	2,3,6	2,6	3
9	9	1,7,8	7	1,8	2	1,7,8	1,7	8

links become inoperative. The inclusion of data distribution in the formulation of  $S$  is the main area of departure of this research from previous work on network survivability. The importance of including the data distribution within a DDP in enumerating survivability is demonstrated in the numerical results presented in the previous section.

The main purpose of this preliminary activity in deriving a DDP survivability criteria was to establish the validity of the index  $S$ . Future plans call for deriving better mathematical representations and improving the computational

algorithms for  $S$ . Assuming the availability of better representations for DDP survivability, it becomes possible to evaluate DDP architectures, data distributions, and to develop algorithms to achieve optimal DDP survivability in terms of these parameters.

The authors want to take this opportunity to express our appreciation for the generous support of this research activity by Mr. Clifford McLain, and Dean Ray, Defense Civil Preparedness Agency; Center for Academic and Administrative Computing; and helpful suggestions by Professor T.

TABLE II.—Value of survivability index  $S$  for various network structures and file distributions

DDP Architecture		Survivability Index $S$		
Figure	Structure Name	Distribution 1	Distribution 2	Distribution 3
3a	Grid	0.748	0.752	0.753
3b	Linear Z	0.393	0.520	0.557
3c	Ring with Tail	0.634	0.603	0.567
3d	Swastika	0.522	0.562	0.576
3e	Star	0.608	0.659	0.690
3f	Ring	0.611	0.625	0.611
3g	Ring with 3 Tails	0.620	0.674	0.657

Lee, and other members of the Electrical Engineering and Computer Science Faculty at The George Washington University.

## REFERENCES

1. Roberts, L. G., et al., "The ARPA Network," *Computer Communications Networks*, pp. 485, Prentice Hall, 1973.
2. Tymes, L., "TYMNET—a Terminal Oriented Communication Network," *AFIPS Conference Proceedings*, Vol. 38, pp. 211, spring, 1971.
3. Frank, H. and Frisch, I. T., "Analysis and Design of Survivable Networks," *IEEE Trans. Comm. Tech.*, Vol. COM-18, pp. 501-519, October 1970.
4. Frank, H. and Frisch, I. T., *Communication, Transmission and Transportation Networks*, Reading, Mass.: Addison-Wesley, 1971.
5. Leggett, J. D., "Synthesis of Reliable Networks," Ph.D. Dissertation, University of Pennsylvania, 1969.
6. Ayoub, J. N. and Frisch, I. T., "Optimally Invulnerable Directed Communication Networks," *IEEE Trans. Comm. Tech.*, Vol. COM-18, pp. 484-489, October 1970.
7. Boesch, F. T. and Feltzer, A. P., "A General Class of Invulnerable Graphs," *Networks*, Vol. 2, pp. 261-283, J. Wiley, 1972.
8. Frank, H., "Vulnerability of Communication Networks," *IEEE Trans. Comm. Tech.*, Vol. COM-15, pp. 778-789, Dec. 1967.
9. Hakimi, S. L. and Amin, A. T., "On the Design of Reliable Networks," *Networks*, Vol. 3, pp. 241-260, J. Wiley, 1973.
10. Erdas, P. and Renyi, A., "On the Evolution of Random Graphs," *Publ. Math Inst. Hung. Acad. Sci.*, Vol. 4, pp. 17-61, 1960.
11. Erdas, P. and Renyi, A., "On the Strength of Connectedness of a Random Graph," *Acta Math.*, pp. 261-267, 1961.
12. Rapoport, A., "Nets with Distance Bias," *Bull. Math. Biophys.*, 13, pp. 85-91, 1951.
13. Rapoport, A., "Contribution to the Theory of Random and Biased Nets," *Bull. Math. Biophys.* 19, pp. 257-277, 1957.
14. Baran, P., "On Distributed Communication Networks," *IEEE Trans. Comm. Tech.*, Vol. COM-12, pp. 1-9, 1964.
15. Brown, D. B., "A Computerized Algorithm for Determining the Reliability of Redundant Configuration," *Trans. on Reliability*, R-20, pp. 121-124, 1971.
16. Hansler, E., McAuliffe, G. K. and Wilkov, R. S., "Exact Calculation of Computer Network Reliability," *Networks*, Vol. 4, pp. 95-112, J. Wiley, 1974.
17. Van Sylke, R. and Frank, H., "Network Reliability Analysis: Part I," *Networks*, Vol. 1, pp. 279-290, J. Wiley, 1972.
18. Barnes, G. H., et al., "The ILLIAC IV Computer," *IEEE Trans. on Comp.*, C-17, pp. 746-57, 1968.
19. Crane, B. A., et al., "PEPE Computer Architecture," *Proceedings of COMPCON*, pp. 57-60, Sept. 1972.

# An operating system kernel mechanism for the poly-processor system PPS-R

by MAKOTO AMAMIYA and NAOHISA TAKAHASHI

*Musashino Electrical Communication Laboratory, N.T.T.  
Musashino, Japan*

and

YUTAKA OGAWA and KENJI KOYAMA

*Yokosuka Electrical Communication Laboratory, N.T.T.  
Yokosuka, Japan*

## INTRODUCTION

Conventional functionally centralized systems have several problems such as cost-performance ratio, reliability, availability, system software productivity, and expandability, for changing the system size and service grade in time sharing systems (TSS).

The functionally decentralized systems, on the other hand, can resolve these problems and recent advances in LSI technology and microprogramming technology enable us to realize such systems. The Poly-processor system (PPS), whose operating system is discussed in this paper, is a computer complex consisting of many small, tightly coupled, functionally dedicated processors. The research version of PPS (PPS-R) has been developed, and now its operating system aiming at TSS services is under development.

Computer complexes such as C.mmp<sup>1</sup>, TIP<sup>2</sup>, MCS<sup>3</sup> and TANDEM 16<sup>4</sup> have been developed, all of which are homogeneous multiprocessor systems and have special purposes.

It is a hard but very important problem to decompose the conventional TSS OS functions and to execute simultaneously each decomposed OS function in order to obtain high cost-performance ratio, system expandability, software productivity, reliability and availability. Few such systems have been realized so far.

The Hyper Operating System Complex for Poly-processor (HOPE-R) is an experimental OS complex whose objective is to resolve the problems mentioned above. HOPE-R is defined as a set of several subOS's which are organized to co-operate to offer TSS services.

This paper introduces the structure and characteristics of PPS-R and HOPE-R, and then discusses the kernel mechanisms of HOPE-R which is the basic frame for realizing HOPE-R service policy and experimental environment. The kernel mechanisms consist of a common kernel mechanism which is required of all subOS's commonly, and specific subOS mechanism which is required of each subOS partic-

ularly, such as I/O control, user-task control, and job-memory control.

## POLY-PROCESSOR SYSTEM

The experimental Poly-processor System (PPS-R) is a computer complex consisting of many small, tightly coupled, functionally dedicated processors, which has been developed for TSS services.<sup>5</sup> PPS-R consists of a processor subsystem, a memory subsystem and a connection subsystem, as shown in Figure 1. The PPS-R hardware configuration and system characteristics are briefly described in the following.

### *Processor subsystem*

The processor subsystem is composed of six functionally dedicated processor classes, in order to increase the system throughput, to shorten the response time and to improve system reliability. The set of functions for each processor class, shown in Table I, is determined corresponding to the partitioning of conventional operating system (OS) functions. Each processor class has one processor.

### *Memory subsystem*

The memory subsystem consists of six memory classes, shown in Table II, which are categorized according to the behavior and characteristics of stored information, in order to provide the function changeability of each processor.<sup>6</sup>

### *Connection subsystem*

The subsystem interconnecting individual hardware components is divided into two classes, InterProcessor Connec-



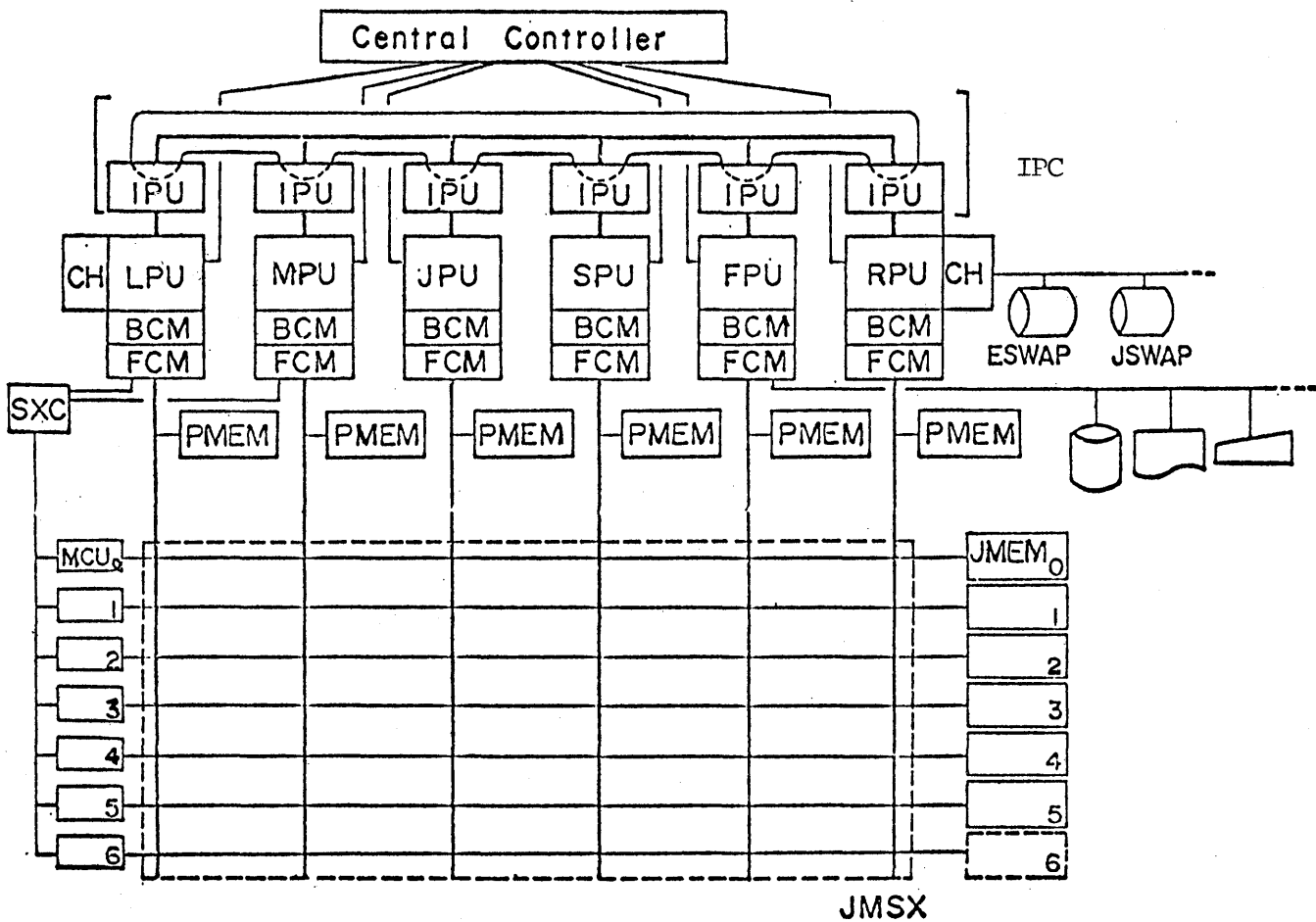


Figure 1—PPS-R organization.

tion (IPC) class and the Job-Memory Switch matrix (JMSX) class. IPC comprises the buses with the controllers (IPU's) which connect processors to each other. The buses contained in this class can transmit information between processors independently at high speed. The JMSX represents the switch matrices which connect each Job Memory (JMEM) with some processors.

**STRUCTURE OF HOPE-R**

HOPE-R functions are classified both horizontally and vertically as shown in Table III.

The six subOS functions are defined according to the following three rules, which are also reflected in each processor function.

- V1: The class on which user programs run must be distinct from the classes on which the EXEC runs.
- V2: The components which have external interaction will be classified into separate classes, depending on the interface processing functions.

- V3: The optional system functions such as the swapping function, which could be selected optionally when the system is installed, must be divided into different classes.

Each subOS function is also classified hierarchically according to the following rules.

- H1: Service dependent functions must be separated from service independent functions.
- H2: Functions which hide hardware peculiarity must be separated from other functions.
- H3: SubOS common functions must be separated from subOS peculiar functions.

In the remainder of this paper, are described HOPE-R specific kernel mechanisms, especially process control, inter-subOS communication control and processor memory (PMEM) control, which are common kernel mechanisms, the task control mechanism peculiar to the Job Processing Unit OS (JPU-OS), and the JMEM management mechanism peculiar to the Roll in/out Processing Unit OS (RPU-OS).

TABLE I.—Processor Subsystem

Class	Functions
JPU	executes application programs activated by the user.
EXEC classes	perform EXEC functions.
LPU	performs network control and communication control functions.
FPU	executes I/O operations, file access control and file management.
SPU	performs system resource scheduling.
RPU	executes swapping operations and virtual space management.
MPU	performs system maintenance and system reconfiguration management.

TABLE II.—Memory Subsystem

Class	Functions
Basic control memory (BCM)	stores microprograms to perform basic functions common to all processors.
Firmware control memory (FCM)	stores microprograms to characterize proper hardware functions for each processor.
Processor memory (PMEM)	stores control information and programs to characterize proper software functions for each processor.
Job memory (JMEM)	stores user's programs and utility programs.
EXEC swapping memory (ESWAP)	stores nonresident portions of the EXEC and backup information.
Job swapping memory (JSWAP)	stores programs and user task data rolled out of JMEM.

PROCESS CONTROL MECHANISM

In the conventional OS the execution of a program can be categorized to non-task, system task, and user task. In HOPE-R, only JPU executes user tasks, and EXEC processors execute messages delivered from other processors (mainly JPU). Considering these execution characteristics, the user task is distinguished from others, and only the user task is defined as *task*, while the system task and non-task are defined as *chore*. The characteristic differences between the task and the chore are shown in Table IV. Though multiprocessing of tasks is under the control of JPU-OS, its

memory allocation and scheduling of the task creation are controlled by other subOS's, and a terminal I/O or a file I/O can be executed concurrently with user programs.

OS supervisor routines are initiated by SuperVisor routine Call (SVC). Some of the supervisor routines are placed in other processor classes; therefore their SVC's, which is named External Supervisor routine Call (ESC), are informed to each processor class through message communication, whose mechanism is described in the next section.

Considering that the chore processing which corresponds to the instantiation of a supervisor routine is relatively small and fixed, its multiprocessing control mechanism is designed

TABLE III.—HOPE-R Configuration

OS level	PU class	JPU OS	LPU OS	FPU OS	MPU OS	RPU OS	SPU OS
Service Policy level		Command execution	Terminal control	Catalog management File management	System reconfiguration	Roll in/out schedule Library management	Job schedule Command schedule Load control
SubOS Mechanism level		Task dispatch	Line buffer control	Access control	Diagnosis	JMEM control Object program control	Task generation Task termination
Common Mechanism level		SVC control	Interrupt handling	Event schedule	Chore control	Load observation	IPC control
		PMEM management	Queueing/dequeueing				
Hardware Virtualizer level			Line control	I/O control	Diagnosis instruction	Channel, I/O schedule Retrial	

TABLE IV.—Characteristics of Task and Chore

	Task	Chore
Processing object	Command from terminal users	Message from other processors
Program	User program, utility program	Exec program
Memory	Job memory	Processor memory
Execution	Full of variety	Restricted and fixed
CPU occupation	In danger of permanent occupation	Sure not to occupy

so as to reduce the control overhead. Another important design point is to organize the software support for emergency communication facility and changing the subOS functions when load imbalance between processors or processor failure occurred.

#### Task control mechanism

The HOPE-R task executes user commands as shown in Figure 2. Characteristic features of the HOPE-R task processing are: (a) The user program runs on JPU, which delivers various demands to other (EXEC) processors through ESC's, (b) RPU manages task areas, independent of the JPU execution, (c) SPU schedules user commands to decide the order of task execution and memory allocation.

##### (1) Task states and state transition

Task states are defined as shown in Figure 3 according to the following rules;

- T1: In what type of storage the task is placed.
- T2: Whether the task requires JPU or not.
- T3: What type of I/O the task is executing, or it has done.

The state transition is shown in Figure 4. Schedule Processing Unit (SPU) status, JPU status, and RPU status are controlled by SPU-OS, JPU-OS, and RPU-OS respectively. The RPU status is considered as an External state in JPU-OS and the JPU status is an External state in RPU-OS.

##### (2) Task dispatching

According to the task characteristics described in Table IV, task dispatching has the following features compared with chore dispatching: (a) the task execution is pre-empted at the end of the time slice; (b) user-exit routines for task interrupts can be set by the user; (c) JPU supervisor routines

manage the task creation, task deletion, return from External state and interruption, etc.

##### (3) Job memory management

RPU-OS manages and allocates JMEM space to the three types of areas: (a) procedure area of user program, (b) data area of user program, (c) task control information area.

RPU-OS manages five states, four of which are defined by rule T1, that is, the states in which the task is placed outside JMEM, and the other is terminal I/O wait state in which the task may be rolled out from JMEM. In PPS-R processors, as base-registers are used for program relocation, instead of paging mechanism, memory space must be allocated in contiguous area. The memory fragmentation problem which occurs in this case, however, can be resolved, because the fragmented areas can be compactified by the RPU-OS specific chore which is independent of JPU-OS task control.

#### Chore control mechanism

In subOS, ESC Messages (ECM's) from other subOS are multiprocessed in order to utilize resources efficiently. The ECM processing control and the chore control flow are shown in Figure 5.

##### (1) Chore control

Chore characteristics in Table IV and software simulation<sup>5</sup> show that the chore execution is small scale and pre-defined fixed type. Considering this fact, chore states are designed as shown in Figure 6.

The chore scheduling is designed so as to reduce the chore control overhead and to obtain subOS independency. The scheduling strategy is as follows: (a) a chore is suspended only when it waits for ECM's or I/O completion, i.e. no time slicing; (b) chores in the ready state should be dispatched before the new chores are created. Then the newly created chore is executed immediately; (c) the emergency ECM should be scheduled at the highest priority.

ECM's and various interruptions, which are called events, are taken up in two methods, look-in method and interrupt method, in order to realize subOS independent chore scheduling and efficient multiprocessing, and to enable the emergency ECM to interrupt the normal chore execution. The result event acceptance rule is shown in Table V.

Also considering the regularity and smallness of the chore processing, chores are designed to wait for only one event, in order to simplify the chore control mechanism.

##### (2) Processor memory management

PMEM space of each processor is allocated for chores. The subOS memory manager allocates and deallocates

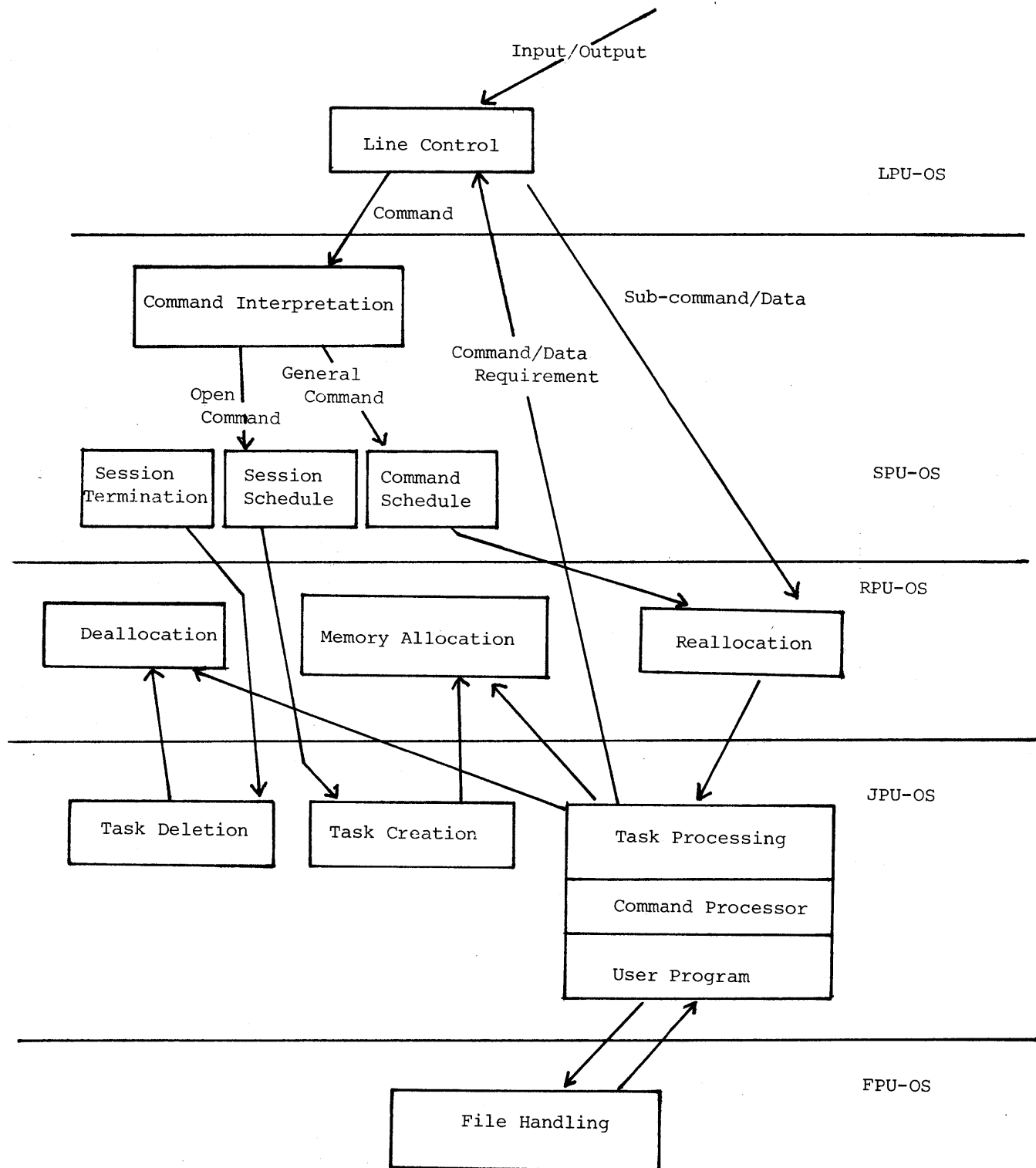


Figure 2—A diagram showing the command processing flow.

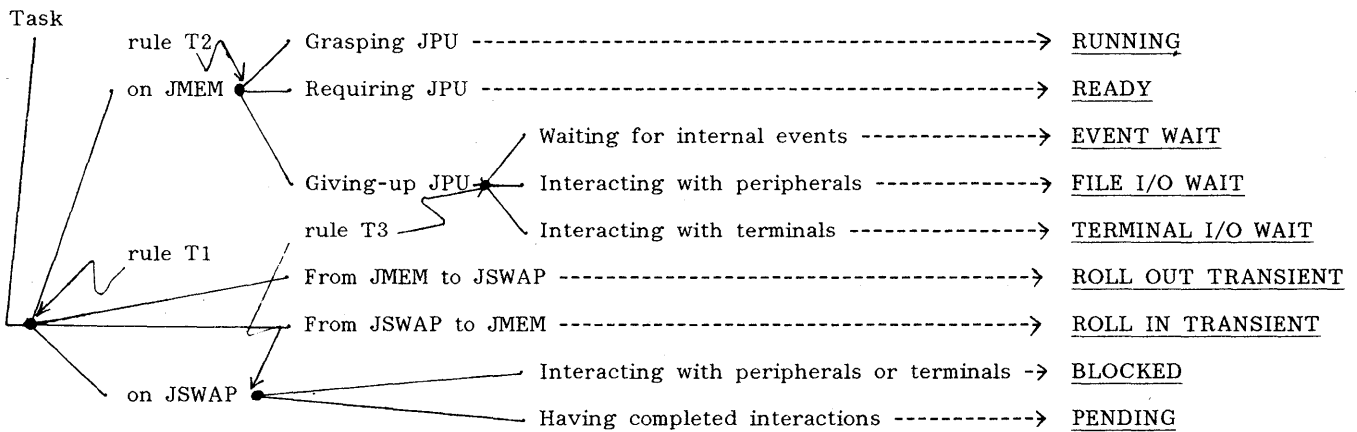


Figure 3—Definition of task states.

PMEM areas to each chore dynamically when they are created and deleted. The chore processing requires fixed sized memory space which is used for chore control blocks, and variable sized memory space which is used for message data and I/O data blocks. Fixed sized memory space is acquired as a block of 16 words, and variable sized memory space is acquired as a block of  $2^n$  ( $n=2,3,\dots,11$ ) words.

**(3) Function changing control**

Function changing facilities are inevitable in the functionally distributed system, in order to resolve the load-imbal-

ance and processor failure. PPS-R hardware has this function changing mechanism as described before. The HOPE-R subOS controls several logical classes of processors. Each subOS manages the chore of all classes one at a time, and the processor class switching is done by dispatching chores selectively in a specific class in order to execute as proxy for the heavy-loaded or failed processor, when an emergency message interruption has occurred.

**INTER-SUBOS COMMUNICATION**

Message data such as ESC's, which are small size but have a high frequency transfer rate, are transmitted by the inter-processor communication system (IPC) of PPS-R<sup>8</sup>. Considering the hardware facilities of data transmission, the inter-subOS communication mechanism is designed to realize the following: (a) the OS programming should be made easy, especially it should be enabled to call all subOS functions without notices to the Poly-processor structures in order that the kernel mechanisms should be free from the change of OS policies or services; (b) errors should be detected at the caller side in order to prevent errors from spreading and avoid unessential communications; (c) it should be enabled to obtain high concurrency between processor classes and to reduce the interprocessor communication overhead, in order to draw out the effect of function distribution.

*SVC mechanism*

Due to the function distribution, some of the SVC's, i.e. ESC's, cause inter-subOS communications.

The HOPE-R SVC mechanism facilitates system programmers to set up kernel or policy routines only by an SVC name without worrying about on which processor such an SVC routine runs.

Though HOPE-R offers two types of ESC linkage, Call/

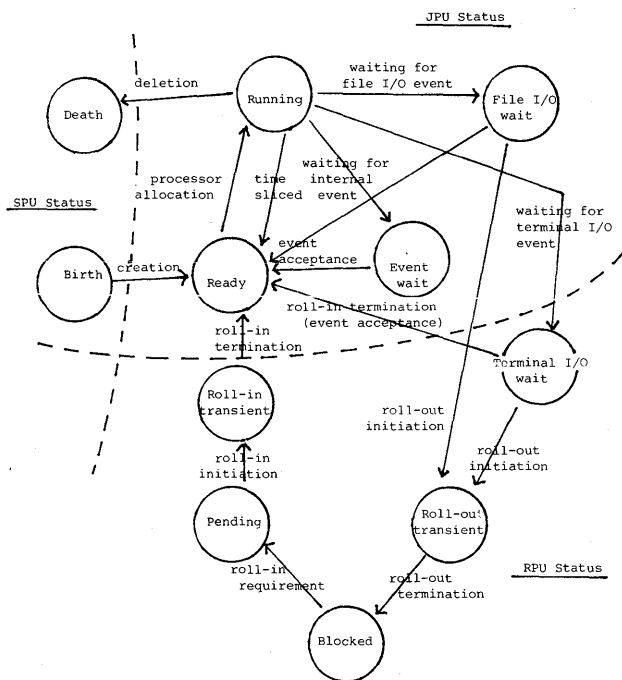


Figure 4—Task state transition diagram.

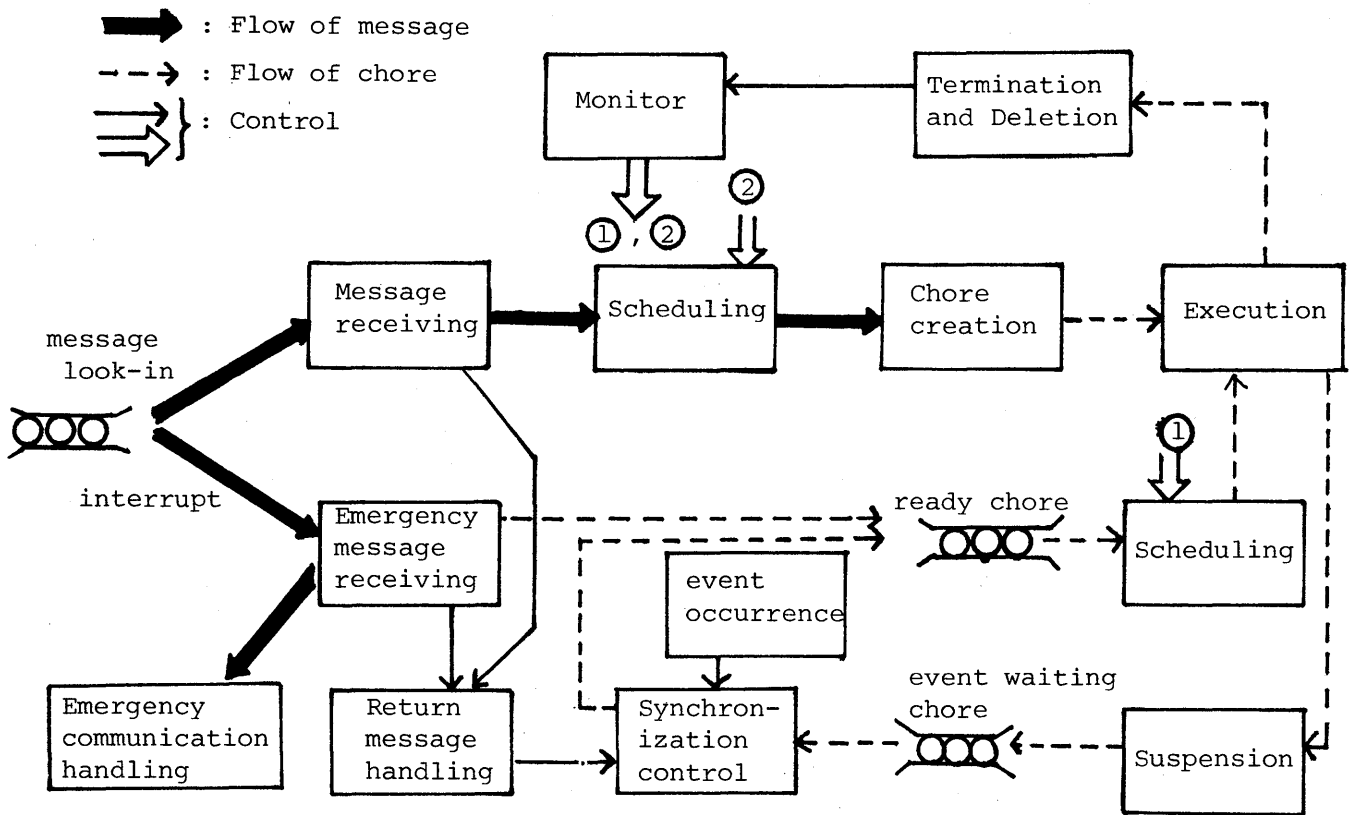


Figure 5—A diagram showing the chore processing flow.

Return type and Transfer type, the Transfer type is preferred from the viewpoint of reducing the communication overhead. For example, Figure 7 shows that the Transfer type linkage takes only five steps in a command requirement processing, while the Call/Return type takes eight steps.

*ESC control mechanism*

ESC control mechanism consists of a call control mechanism, an SVC body execution and a return control mechanism.

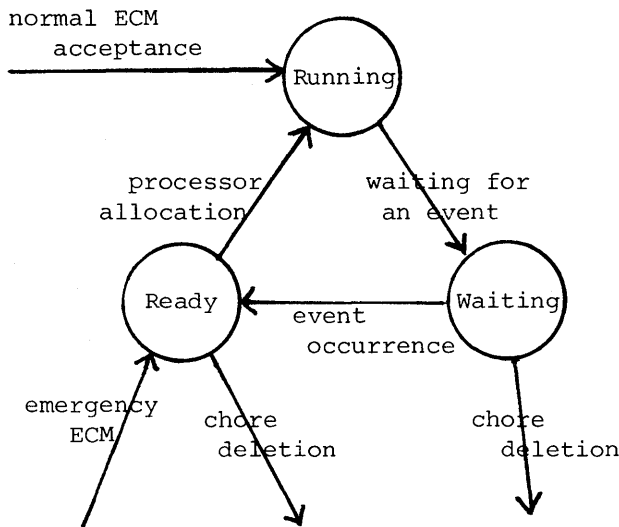


Figure 6—Chore state transition diagram.

TABLE V.—Event Acceptance Rule

Event class	Conditions of acceptance	Acceptance method	Priority	Processing on acceptance
Emergency ECM	Event occurrence	Interrupt	1	Emergency chore creation
Normal ECM	No "ready" chore and no ERM	Look-in	5	Normal chore creation
ERM (ESC return message)	No "ready" chore	Look-in	4	
Message of I/O completion	Event occurrence	Interrupt	2	Sending messages to chores
Timer	Dispatching	Look-in	3	
Chore created message	Event occurrence		2	

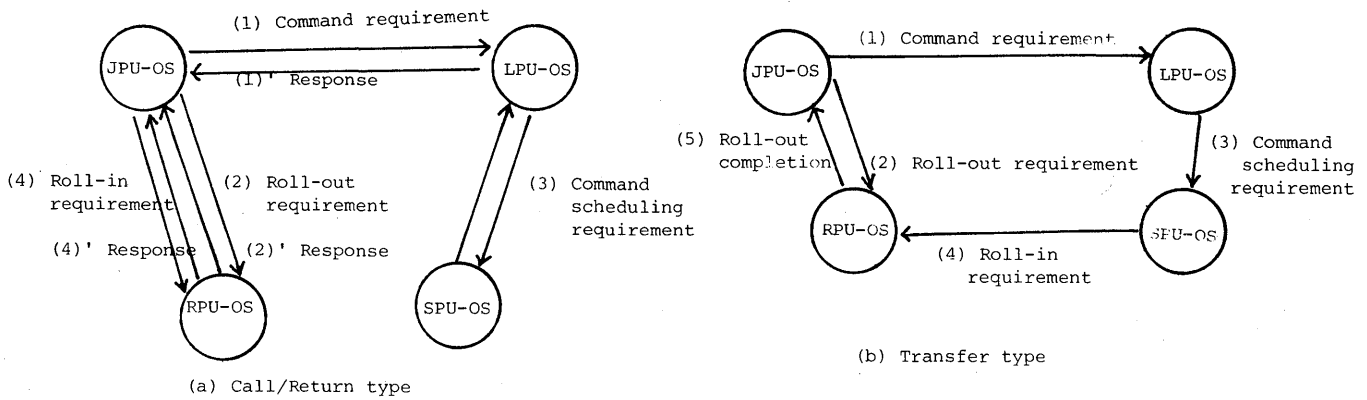


Figure 7—Two types of ESC linkage in command processing.

The call controller translates SVC name and parameters to a message (ECM) form referring to the SVC table in which the translation information is written, and then sends the message to the destination subOS. In this process, data errors are checked out. When the destination subOS receives the message, it sets up an execution environment of the ESC from ECM information, then calls the SVC routine. In the case of Call/Return type, on returning from the SVC, the return controller translates the return parameters to a return message and sends the message to the caller subOS. The caller subOS, when he receives the return message, reconstructs the return parameters and returns them to the caller routines. In the case of Transfer type, the return controller does nothing but the SVC body delivers an ESC with return parameters toward the other subOS.

High Level (HL) protocol transmits an ESC message, using Message PUT (MPUT) and Message GET (MGET) primitives. MPUT divides the message to several blocks and passes them to Remote WRITE (RWRITE) primitive of Low Level (LL) protocol. MGET reconstructs a message from blocked data passed by Remote READ (RREAD) primitive of LL protocol. LL protocol transmits blocked data up to 32 words, which is called Sending Control Words (SCW). SCW includes control information which is used to set up the communication link. Basic Level (BL) protocol is the PPS-R hardware interface.

The protocol processing flow is summarized in Figure 8.

*Inter-subOS communication protocol*

The protocol hierarchy is set as shown in Table VI, considering the correspondence to the HOPE-R function hierarchy.

*Emergency communication*

Anomalies in execution are informed by emergency communication (EC). Emergency messages are transmitted by EC primitives which are shown in Table VII. EC primitives are privileged to use IPC, and the emergency message interrupts the destination processor.

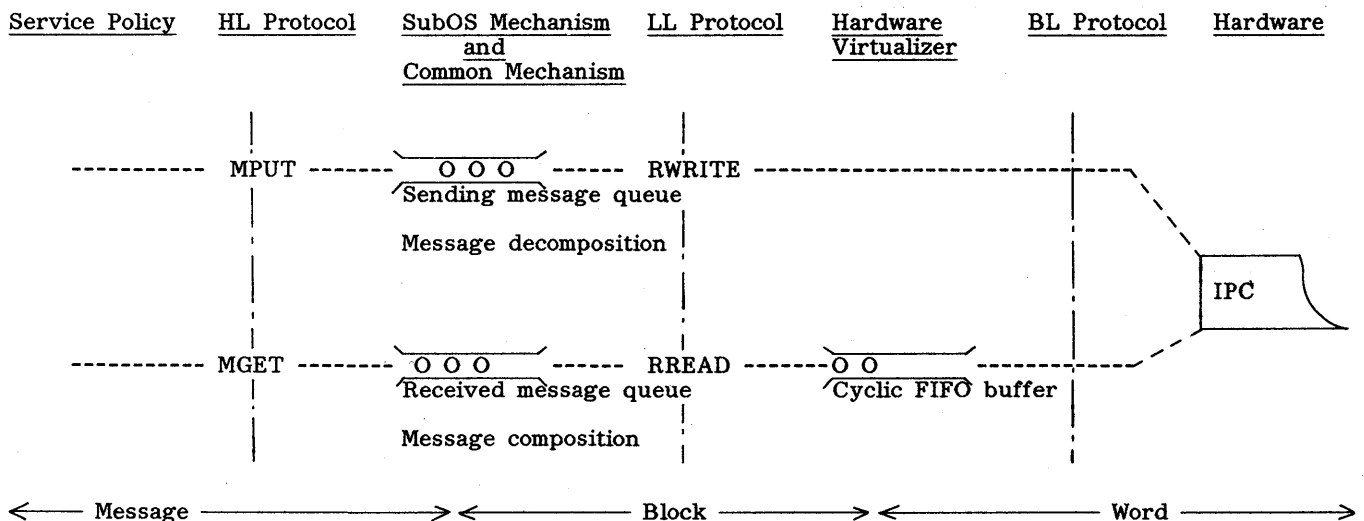


Figure 8—A diagram showing the inter-subOS communication flow.

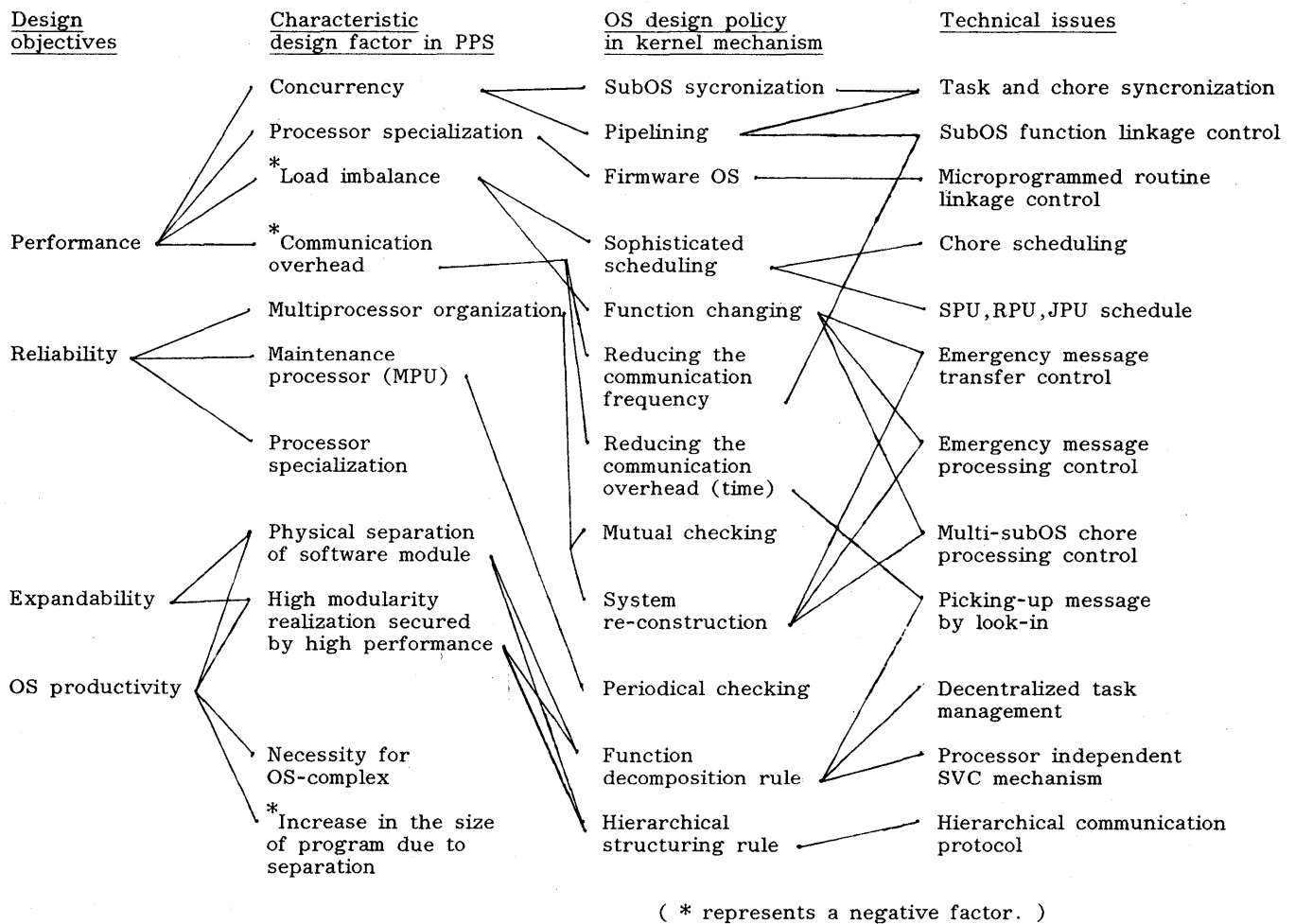


Figure 9—Design objectives and kernel mechanisms.

DISCUSSION

HOPE-R kernel mechanism makes an offer the experimental environment to realize TSS service policies on poly-processor system. The reflections of Poly-processor re-

search objectives to PPS-R and HOPE-R kernel mechanism features are shown in Figure 9. Among the issues, the important problems are discussed in the following.

- (1) The chore concept, which is newly introduced to the

TABLE VI.—Inter -subOS Communication Protocol

Hierarchies		Function	Primitives
Protocol level	OS level		
High level	Common mechanism	Message transfer	MPUT MGET
Low level	Hardware virtualizer	Block transfer ( block length is variable up to 32 words )	RWRITE RREAD
Basic level		Word transfer ( 1 word = 16 bits )	

TABLE VII.—Emergency Communication Primitives

Item	Contents
Emergency state occurrence	Declaring an emergency state and requiring to suppress any communications on the bus at once ( via broadcast line )
Emergency interrupt	Interrupting normal processors to make their state emergency
Emergency saving	Transferring information, stored in the faulty processor's PMEM, to a normal processor
Emergency processor down	Informing normal processors that the faulty processor has fallen into logical down state
Emergency state termination	Removing the bus use suppression ( via broadcast line )



processing of message in HOPE-R, unites a logical processing frame and an actual control frame and makes it easy for a programmer to design SVC routines. From the efficiency point of view, chore control mechanisms such as creation, deletion, synchronization, are much simpler than the general TSS process control mechanism such as in MULTICS.

- (2) HOPE-R kernel has two types of inter-subOS function control linkage, Call/Return type and Transfer type. Among the two, the latter is considered the necessary method in the functionally distributed multiprocessor system. The merit of this method is to shorten the number of communications in a processing sequence, and to reduce the linkage overhead.
- (3) HOPE-R chores wait for only one event. This simplifies the chore control. As PPS-R processors execute concurrently many tasks, high throughput will be obtained by pipelined effect even in that simplified control method.
- (4) PPS-R can detect processor failure by processor self checking and periodical checking by Maintenance Processing Unit (MPU). In order to obtain high reliability, it is necessary to detach the failed processor and to reconfigure the system with remaining elements.<sup>8</sup> The emergency communication control mechanism is important to realize those facilities in HOPE-R.
- (5) The concurrency between the task execution in JPU and the task area management in RPU is effective to obtain higher throughput and to shorten the response time.
- (6) The interprocessor communication overhead is a weak point of distributed systems. In HOPE-R, this communication overhead is serious in compensation for the flexibility of the SVC mechanism. However as the communication control is the fixed processing, its routines can be made by firmware or set in the IPC hard-

ware device. Furthermore the memory management for message data blocks, which shows a high overhead percentage, can be simplified by setting the exclusive memory pool for message data.

HOPE-R TSS policy routines are now under development with the aims to evaluate the feasibility of Poly-processor system and the HOPE-R design philosophy described in this paper.

#### ACKNOWLEDGMENTS

The authors wish to thank to Dr. K. Murakami, who was the PPS project leader, for his helpful guidance, and PPS project members for fruitful discussions. They also wish to thank Mr. K. Yamashita, the director of their section, for giving them an opportunity to write this paper.

#### REFERENCES

- 1 Wulf, W. A. and Bell, C. G., "C.mmp A multi-mini-processor," *Proc. of FJCC*, pp. 765-777, 1972.
- 2 Heart, F. E., et al., "A New Minicomputer/Multicomputer for the ARPA Network," *NNC*, pp. 529-537, 1973.
- 3 Jensen, E. D., "A Distributed Function Computer for Real Time Control," *Proc. of 2nd Annual Symposium on Computer Architecture*, pp. 176-182, 1975.
- 4 TANDEM 16 System Introduction, TANDEM Computer Inc.
- 5 Murakami, K., et al., "Poly-Processor System Analysis and Design," *Proc. of 4th Annual Symposium on Computer Architecture*, pp. 49-56, 1977.
- 6 Sato, M., et al., "Dynamic Function Exchanging Mechanism in Poly-Processor System," *Proc. of 6th Annual Symposium on Computer Architecture*, pp. 130-136, 1979.
- 7 Nishikawa, S., et al., "Inter-connection Unit for Poly-Processor System: Analysis and Design," *Proc. of 5th Annual Symposium on Computer Architecture*, pp. 216-222, 1978.
- 8 Takahira, S., et al., "A Reliability Aspect of Function Distribution System: PPS," *Proc. of 3rd UJCC*, pp. 70-74, 1978.

# Measures for distributed processing network survivability

by GENE HILBORN

Ford Aerospace and Communications Corporation  
Palo Alto, California

## INTRODUCTION

We seek to develop a new methodology for describing and comparing the survival effectiveness of connected networks of processing subsystems (termed "distributed processing networks") which are subject to deliberate or natural failures or losses of processing or communication components. Examples of distributed processing networks include:

- resource-sharing computer networks
- military force teams
- distributed architecture computers
- distributed sensor-processing systems
- distributed ballistic missile defense systems

The key idea which underlies the approach taken in this paper is that of a *team* system which depends both on the existence and communication connectivity of its member components for full performance. We then describe the attributes of existence and connectedness of the processing subsystems as the nodes and edges of a graph, and define a performance index axiomatically on the connectivity state space of the graph. To the extent that this performance captures the essentials of the "team effect" it allows survivability cost/performance trades of alternate network architectures.

While key results are expressed concisely as theorems, formal proofs are omitted.

## GRAPH THEORY AND BACKGROUND

### Definitions

In order to present further details of the theory, it is necessary to introduce certain graph theory terminology.

A *graph* is a set of *nodes* (representing processing subsystems connected by *edges* (arcs, lines) (representing communications links). If the edges are directed with arrows (representing, for example, one-way message flow), the graph is technically called a *digraph*. We will be concerned only with *undirected* graphs, comprised of nodes and un-oriented edges.

A graph is *connected* if there is a path between every distinct pair of nodes.

The edges connected to a node are said to be *incident* at the node. The number of edges incident to a node is called the *degree* of the node.

The set of nodes with edges connected to a given node  $n_i$  is said to be *adjacent* to  $n_i$ .

A graph in which all other nodes are adjacent to each node is called *complete* (or completely connected).

### Notation and vulnerability background

Let  $G=(N,E)$  denote a graph comprised of a non-empty set of  $n$  nodes,  $N$ , and set of  $e$  edges  $E$ . We consider only distinct, non-oriented edges between distinct nodes  $n_i$  and  $n_j$  denoted  $[n_i,n_j]$  or  $e_{ij}$ . Parallel edges between a pair of nodes and self loop edges  $[n_i,n_j]$  will not be considered. The set of nodes adjacent to node  $n_i$  is denoted by  $\Gamma(n_i)$ . If  $|\cdot|$  denotes the size of a finite set, the degree  $d_i$  of node  $n_i$  is given by

$$d_i = |\Gamma(n_i)|.$$

The total of node degrees counts each edge twice. Hence

$$\sum_1^n d_i = 2e.$$

If all nodes are of equal degree, then

$$d = 2e/n,$$

and the graph is called *homogeneous* or *regular* of degree  $d$ .

Previous work on network vulnerability has concentrated on the difficulty of disconnecting a connected graph.\* A set of edges whose removal disconnects a connected graph is called an edge disconnecting set. The size of the smallest edge disconnecting set is called the *cohesion*— $\lambda(G)$ . A node disconnecting set is a set of nodes whose removal disconnects a connected graph. The size of the smallest node disconnecting set is called the *connectivity*— $\omega(G)$ .  $\omega(G)$  for a complete graph is defined to be  $n-1$ .

The basic result for the cohesion/connectivity property of graphs\*\* is that the minimum number of nodes that can dis-

\* A complete summary of results on vulnerability is given in a survey paper by Wilkor.<sup>1</sup>

\*\* A proof is given by Frank and Frisch.<sup>2</sup>

connect a graph is never larger than the number of edges to disconnect the graph:

$$\omega \leq \lambda \leq d_i, \quad 1 \leq i \leq n.$$

Also, since  $\lambda \leq d_{\min} \leq 2e/n$ , a graph least vulnerable to disconnection (for a given number of edges and nodes) which could exist would give

$$\omega = \lambda = \lfloor 2e/n \rfloor,$$

where  $\lfloor \cdot \rfloor$  denotes truncation to integer part. Moreover, such graphs do always exist.

Cohesion and connectivity are clearly of some value in comparing alternate distributed processing network topologies for survivability under link and node failure or attack, respectively. However, we are interested in a more generalized measure which can accommodate any amount of loss and, more importantly, can usefully measure effectiveness beyond simple connectivity—disruption.

### AN AXIOMATIC SURVIVAL INDEX

#### Definition

The graph  $G=(N,E)$  of the distributed processing network before any element is removed (by attack or failure) is connected. Removal of some elements of  $G$  creates a *reduced graph*  $G'$ . The reduced graph contains  $\ell$  isolated subgraphs, where each subgraph is connected. (These subgraphs are also called the *component subgraphs*.) More precisely, if  $G'=(N',E')$  is a reduction of  $G$ , then

$$G' = \bigcup_1^{\ell} G_i' = \left( \bigcup_1^{\ell} N_i', \bigcup_1^{\ell} E_i' \right),$$

where each component subgraph  $G_i'$  is connected and no path exists between any nodes, in distinct sets  $N_i'$  and  $N_j'$ .

We now define a connectivity state vector  $x$  on any reduction  $G'$  of  $G$  as the  $n$ -vector with components equal to the size of respective component subgraphs:

$$x_i = |N_i'|, \quad i \in I_n = \{1, 2, \dots, n\}.$$

However, any re-indexing of the state space components does not reflect any change in connectivity or function—all such permutations being equivalent. As a convenience, this equivalence can be formalized by adopting the convention that the subgraphs are indexed such that  $|N_1'| > |N_2'|$ , etc. Then

$$n \geq x_1 \geq x_2 \geq \dots \geq x_{\ell} > 0.$$

Let the space of  $x$  thus defined be called  $\Omega_n$ . While the space  $I_n$  has  $n^n$  points,  $\Omega_n$  is smaller but still rapidly growing with  $n$ . This combinatorial reduction is illustrated in the partial

table below:

$n$	$n^n$	$ \Omega_n $
1	1	1
2	4	3
3	27	6
4	3.1E3	11
5	8.2E5	18
.....	.....	.....
10	1.0E10	142
.....	.....	.....
20	1.0E26	2.7E3

Some special connectivity states will be useful to define:

$$x_c = (n, 0, \dots, 0)$$

$$= x \in \Omega_n \text{ for any connected } G=(N,E)$$

$$x_f = (1, \dots, 1)$$

$$= x \in \Omega_n \text{ for completely disconnected set of } n \text{ nodes } G'=(N, \emptyset)$$

$$0 = (0, \dots, 0)$$

$$= x \in \Omega_n \text{ for empty graph } G'=(\emptyset, \emptyset).$$

We are now in a position to define a *survival index* as a function on  $\Omega_n$  onto the unit interval  $[0,1]$  satisfying the following axioms:

$$A0) \sigma: \Omega_n \rightarrow [0,1]$$

$$A1) \sigma(0) = 0$$

$$A2) \sigma(x_c) = 1$$

$$A3) \sigma(x) < 1 \Rightarrow \Delta_i \sigma(x) > 0, \quad i \in I_{\ell+1}$$

$$A4) x_i > x_j \Rightarrow \Delta_i \sigma(x) > \Delta_j \sigma(x), \quad i, j \in I_{\ell+1}.$$

The notation " $\Delta_i \sigma(x)$ " refers to a partial difference or increment:

$$\Delta_i \sigma(x) = \sigma(x + \delta_i) - \sigma(x), \quad \text{for } x, x + \delta_i \in \Omega_n$$

where  $\delta_i$  is an  $n \times 1$  unit vector with 1 in the  $i$ th position.

Axioms A0-A2 are conventions. Axioms A3 and A4 have the following interpretations/motivations.

Axiom A3 means that adding a node to the graph always improves system performance.

Axiom A4 further orders the connectivity state space by asserting that performance is improved more by adding a node to the larger of the different connected subgraphs. This also includes the empty subgraph ( $x_{\ell+1} = 0$ ); thus, in particular, adding a node to an existing connected subgraph is better than adding it as an isolated node. Axiom A4 formalizes the notion of a "team effect" or synergy effect.

The defining of  $\sigma(\cdot)$  as a function on the space of the connectivity state is equivalent to the assumption that all team members are of equal importance. This homogeneity is the ideal in decentralized systems—no member is more vulnerable or important to the overall system than any other.

#### Shaping

If we have any function  $f(\cdot)$  on the state space which is a survival index (i.e., satisfies A0-A4), we may wish to mod-

ify or *shape* it to model the behavior of a class of proposed systems. The following theorem allows us to do this and still preserve the axiomatic properties of our function.

*Theorem* If  $f(\cdot)$  is a survival index function and  $g(\cdot): [0,1] \rightarrow [0,1]$  is monotone increasing with  $g(0)=0$  and  $g(1)=1$ , then the composite function  $g \circ f$  is also a survival index.

The inner function  $f$  above will be called the *core* function, and the outer function  $g$  the *shaping* function.

### Ordering and bounds

The axiomatic definition of the survival index has been constructed to produce a partial ordering of the state space  $\Omega_n$ —without assigning values. To illustrate this ordering and judge its reasonableness let  $m = \sum x_i$ . All  $x \in \Omega_n$  for each  $m$  are completely ordered. For example, if  $m=4$

$$\sigma(4) > \sigma(3,1) > \sigma(2,2) > \sigma(2,1,1) > \sigma(1,1,1,1).$$

(For brevity, trailing zeros are dropped in the above notation.)

The basis for this ordering can be stated as follows: Let  $\tilde{\Omega}_m = \Omega_m - \Omega_{m-1}$ . That is,  $\tilde{\Omega}_m$  is the subset of  $\Omega_n (n \geq m)$  for which  $x_i = m$ . Let  $\ell(X)$  denote the number of connected subgraphs of  $X$ . Thus, we have:

*Theorem*  $\tilde{\Omega}_m$  is ordered as follows: For any distinct  $x, y \in \tilde{\Omega}_m$ , with  $\ell(x) \leq \ell(y)$ , then

- (a)  $\ell(x) < \ell(y) \Rightarrow \sigma(x) > \sigma(y)$
- (b)  $\ell(x) = \ell(y) \Rightarrow$  for min  $i$  such that  $x_i > y_i$ ,  $\sigma(x) > \sigma(y)$ .

Since for every distinct  $x, y \in \tilde{\Omega}_m$  either  $\ell(y) = \ell(x)$  or  $\ell(x) \neq \ell(y)$ ; thus the theorem completely orders  $\tilde{\Omega}_m$ .

This ordering provides us with *bounds* on  $\sigma(x)$ , for any graph  $G$  with  $n$  nodes and its reduced graph  $G'$  with  $m$  nodes. Let  $x_c(m)$  denote the connectivity state vector for a subgraph which is connected and of size  $m$ , and  $x_i(m)$  denote the connectivity state for a subgraph consisting of  $m$  isolated nodes. That is,

$$x_c(m) = (m) \quad \text{and}$$

$$x_i(m) = (1, 1, \dots, 1).$$

The bounding result is thus stated:

*Theorem* For any connected graph  $G$ , and any reduced subgraph with connectivity state  $x$  with  $\sum x_i = m$ ,

$$\sigma(x_i(m)) \leq \sigma(x) \leq \sigma(x_c(m)).$$

These bounds follow directly from Theorem 3.3(a). Since  $\ell(x_c(m)) = m$  is maximal,  $\sigma(x_i(m))$  is minimal, and since  $\ell(x_i(m)) = 1$  is minimal,  $\sigma(x_c(m))$  is maximal.

Our choice of axioms also gives reasonable orderings between states of different  $m$ . For example,

$$\sigma(5) > \sigma(4), \sigma(4,1) > \sigma(4), \sigma(3,2) > \sigma(3,1).$$

The space is not completely ordered. For example,  $\sigma(4)$  and  $\sigma(3,2)$  are not ordered by the axioms. This ordering depends *quantitatively* on the strength of the “team effect.”

For example, if we take

$$\sigma(x) = \sum_{i=1}^{\ell} X_i^{\alpha} / n^{\alpha}, \quad \alpha > 1$$

which satisfies the axioms,

$$\sigma(4) \geq \sigma(3,2) \quad \text{according to} \quad \alpha \geq 1.5071.$$

## ATTACKS AND FAILURES

If  $G=(N,E)$  is the original connected network graph, reduced graphs are generated in two ways or combinations:

- Node removals
- Edge removals.

These “removals” could be from successful attacks or from failures.

### Node attacks

A node attack of size  $k$ , denoted  $A_k$ , removes  $k$  nodes from  $N$  (and the associated incident edges).  $A_k$  is the set of  $k$  nodes in  $N$ . For any node attack  $A_k$ , there is a corresponding connectivity state vector  $x(A_k)$ . Let  $\Gamma_k$  denote the set of state vectors resulting from all possible node attack of size  $k$  against  $G$ . Then an *optimal* node attack  $A_k^*$  is defined as any node attack of size  $k$  such that if  $x_k^*$  is the corresponding state in  $\Gamma_k$ , we have

$$\sigma(x_k^*) \leq \sigma(x) \quad \text{for any } x \in \Gamma_k.$$

A performance function under optimal node attack is called the *node* survival function,  $\mu_k$ ,

$$\mu_k = \sigma(x_k^*) = \min_{x \in \Gamma_k} \sigma(x).$$

The question could be raised for whether  $x^*$  is defined solely on the basis of the axioms for  $\sigma$ . The answer is in the affirmative since for each  $x \in \Gamma_k$  we have

$$\sum x_i = n - k.$$

Hence,

$$\Gamma_k \subset \tilde{\Omega}_{n-k}$$

which is uniquely ordered by the Ordering Theorem. Moreover, the Bounding Theorem provides bounds on  $\mu_k$  over all possible edge sets and node attacks of size  $k$ ;

$$\sigma(x_i(n-k)) \leq \mu_k \leq \sigma(x_c(n-k)).$$

Figure 1 illustrates the general behavior of  $\mu_k$ .  $\mu_k$  follows along the upper bound (where all remaining nodes remain connected) until a value of  $k$  is reached where the remaining graph becomes disconnected. It then drops monotonically toward the lower bound, and reaches it when all remaining nodes are isolated. The value of  $k$  for which  $\mu_k$  is first less than the upper bound is precisely the connectivity  $\omega(G)$  or

Node Survival Function

$\mu_k$

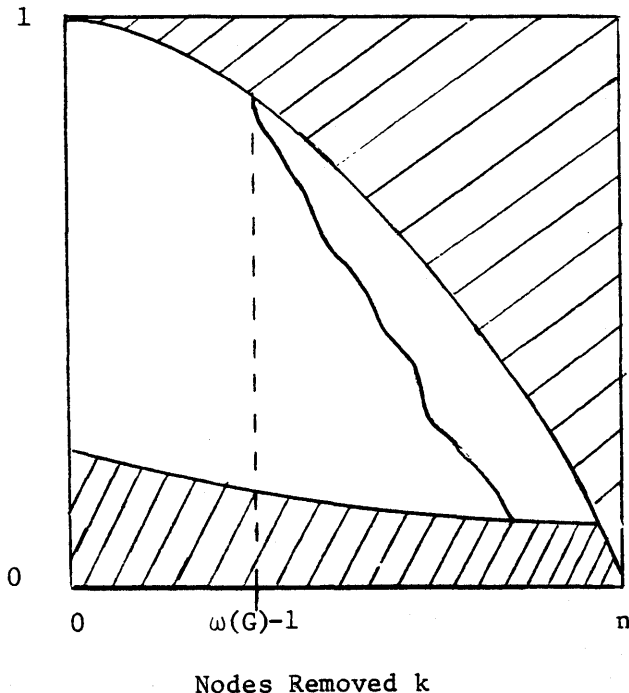


Figure 1—General behavior of the node survival function.

size of the smallest node disconnecting set which has previously been used in investigations of network vulnerability.

An alternative measure to the node survival function is to use *random* node attacks or failures of size  $k$ . By selecting at random any set of  $k$  nodes  $A$ , and the corresponding  $x(A_k) \in \Gamma_k$ . This may correspond more realistically to random attack leakage or natural failure. Thus, we define the *mean node survival function* as

$$\begin{aligned} \bar{\mu}_k &= E\{\sigma(x) | x \in \Gamma_k\} \\ &= \int \sigma(x(A_k)) dF(A_k). \end{aligned}$$

Since  $\mu_k$  is minimal over the same  $\Gamma_k$  as  $\bar{\mu}_k$  we have the bounds:

$$\mu_k \leq \bar{\mu}_k \leq \sigma(x_c(n-k)).$$

Link attacks

Analogous characterization can be made for attacks on communication links of the network (edges of the graph).

A *link attack of size k*, denoted  $B_k$ , removes  $k$  edges from  $E$ . For any link attack  $B_k$  there is a corresponding connectivity state  $x(B_k)$ . Let  $\Delta_k$  denote the set of state vectors resulting from all possible link attacks of size  $k$  against  $G$ . An

*optimal link attack*  $B_k^+$  of size  $k$  is defined as any link attack of size  $k$  such that if  $x_k^+$  is the corresponding state in  $\Delta_k$  we have

$$\sigma(x_k^+) \leq \sigma(x) \quad \text{for any } x \in \Delta_k.$$

A survival function under optimal link attack is then defined as

$$v_k = \sigma(x_k^+) = \min_{x \in \Delta_k} \sigma(x).$$

Since every link attack leaves  $n$  nodes,  $\sum x_i = n$  or  $\Delta_k \subset \bar{C}\bar{Q}_n$ . Thus each  $\Delta_k$  is uniquely ordered by the axioms and  $x_k^+$  is well defined. Similarly, bounds over all possible edge sets and link attacks are:

$$\sigma(x_c(n)) \leq v_k \leq \sigma(x_c(n)) = 1.$$

The upper bound can be made somewhat tighter. The number of edges removed cannot exceed  $e$ , and  $e$  cannot exceed  $n(n-1)/2$  where the graph is complete. For  $k \leq (n-2)(n-1)/2$  there could be enough links to make a complete graph. Thus, the upper bound = 1 for  $k \leq (n-2)(n-1)/2$ . Let  $k_1$  denote  $(n-2)(n-1)/2$  and  $k_2 = n(n-1)/2$ .

For  $k_1 < k \leq k_2$ , we have

$$\sigma(x) \leq \sigma(n-k+k_1, 1, 1, \dots, 1) \quad \underbrace{\hspace{2cm}}_{k-k_1}$$

Note that at  $k=k_2$ , the upper and lower bounds meet. Let  $u.b.(k)$  denote this tighter upper bound. Then

$$\sigma(x_c(n)) \leq v_k \leq u.b.(k).$$

This general behavior of  $v_k$  is illustrated in Figure 2. The smallest value of  $k$  for which  $v_k$  is less than 1 is precisely the cohesion  $\lambda(G)$  defined as the size of the smallest edge disconnecting set.

A link attack performance measure under random link attack/failure can be defined by selecting at random any set of  $k$  edges  $B_k$  with corresponding  $x(B_k)$  in  $\Delta_k$ . The *mean link survival function* (under optimal link attack) is defined as

$$\begin{aligned} \bar{v}_k &= E\{\sigma(x) | x \in \Delta_k\} \\ &= \int \sigma(x(B_k)) F(B_k). \end{aligned}$$

Since  $v_k$  is minimal over  $\Delta_k$ , we have

$$v_k \leq \bar{v}_k \leq u.b.(k).$$

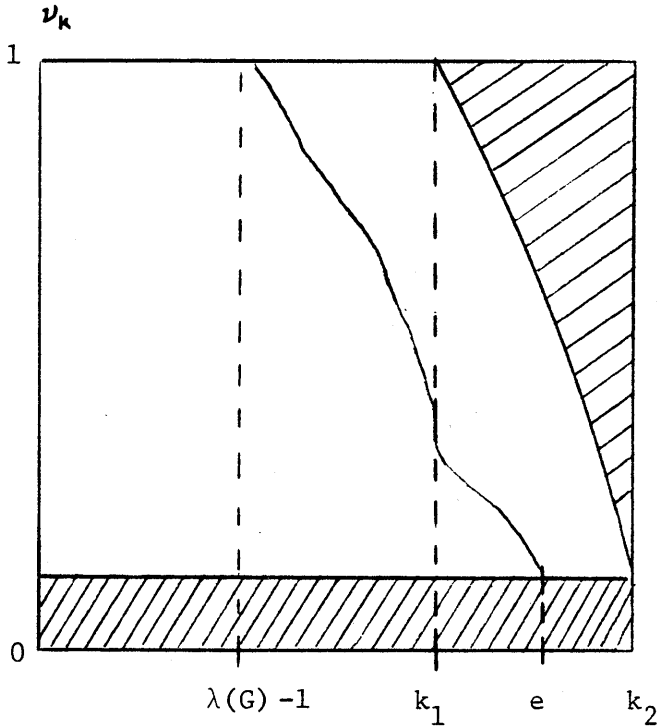
EXAMPLES

A particular survival index

To see how a survival index for a system could be derived independently of our axioms and then tested against the axioms consider a system with distributed resources such that each node— $i$  does work fraction  $w_i$  and the total work  $w$  is

$$w = \sum_1^n w_i.$$

Link Survival Function



Edges Removed  $k$

Figure 2—General behavior of the link survival function.

If resources are evenly distributed among nodes, each node could be modeled as able to do work

$$w_i = \frac{1}{n} \cdot \frac{m_i}{n}$$

where  $m_i$  is the number of nodes whose resources are available to node  $i$  (including its own resources). If node  $i$  is removed  $m_i=0$ ; if it exists  $m_i=1 +$  number of other nodes connected to node  $i$ . Thus, if the graph  $G$  of all  $n$  nodes is connected,  $w=1$ .

Now consider any reduced graph  $G'$  with connectivity state  $x$ . Then

$$\begin{aligned} m_i &= x_1 & \text{for } i=1,2,\dots,x_1 \\ x_2 & & \text{for } i=x_1+1, x_1+2,\dots,x_1+x_2 \\ & \vdots & \\ x_\ell & \text{for } i = \sum_1^{\ell-1} x_i, 1 + \sum_1^{\ell-1} x_i, \dots, \sum_1^{\ell} x_i = m \\ & \vdots & \\ 0 & \text{for } i > \ell \end{aligned}$$

It follows that

$$w(x) = \sum_1^n x_i^2/n^2.$$

This function clearly satisfies axioms A0-A2. To test A3 and A4, note that

$$\Delta_i w(x) = (2x_i + 1)/n^2, \quad 1 \leq i \leq \ell + 1$$

Thus,

$$\Delta_i w(x) > 0 \quad (\text{Axiom A3})$$

and

$$x_i > x_j \Rightarrow \Delta_i w(x) > \Delta_j w(x) \quad (\text{Axiom A4}).$$

A generalization

The preceding survival index can be generalized to the one-parameter class of functions

$$f(x) = \sum_1^n x_i^\alpha/n^\alpha$$

for any  $\alpha > 1$ . Again the axioms can be shown to hold for this function.

If the above function is used as a core function, a second generalization can be made by using the "natural" shaping function  $(\cdot)^{1/\beta}$  for any  $\beta > 0$ . Then we get a two parameter class of survival functions

$$\sigma(x) = \left[ \sum_1^n x_i^\alpha/n^\alpha \right]^{1/\beta}.$$

Now consider the upper and lower bounds for this  $\sigma(\cdot)$  under node attacks:

$$\text{Upper: } \sigma(x, (n-k)) = \left( \frac{n-k}{n} \right)^{\alpha/\beta}$$

$$\text{Lower: } \sigma(x, (n-k)) = \left( \frac{n-k}{n^\alpha} \right)^{1/\beta}.$$

The behavior of the upper bound is illustrated in Figure 3. The convex characteristic for  $\beta > \alpha$  models systems of "robust" performance degradation—with node loss. The concave characteristic for  $\beta < \alpha$  models systems displaying "tender" performance degradation with node loss (while connectivity is maintained). It is important to realize, however, that the ranking of different topologies will be unaffected by  $\beta$ .  $\beta$  only distorts the numerical scale.

16-node network example

Consider the problem of comparing survivability and cost of alternative topologies for a 16-node network, where the nodes are arrayed in a regular  $4 \times 4$  grid. We suppose that the cost per unit distance for edges (communication cable) is one unit between nodes which are vertical or horizontal

Upper Bound

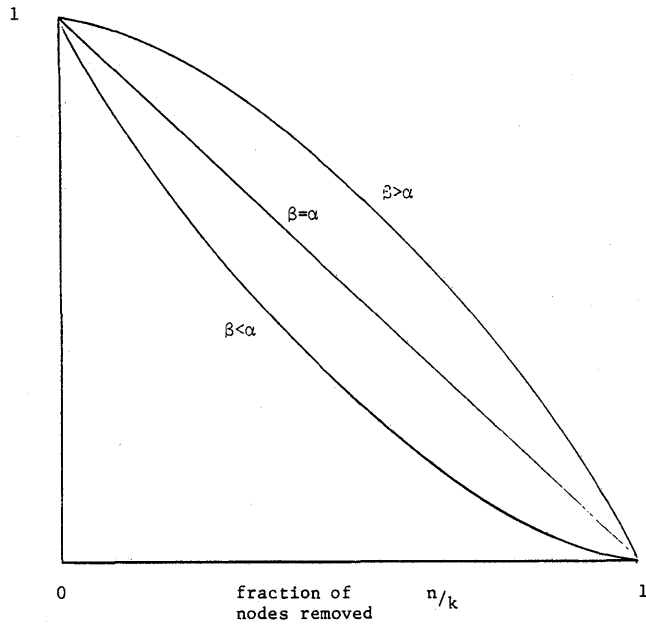


Figure 3—Upper bound or survival index for the two parameter example function under node attack.

TOPOLOGY	CONNECTIVITY ( $\omega$ )	COST
Star Tree	1	15
String Tree	1	15
Loop	2	16
Grid	2	24
X-Grid	3	$\approx 33$
4-Grid	4	$\approx 40$
Complete Graph	15	$\approx 270$

Figure 4—16-node alternate topology examples.

Node Survival Function

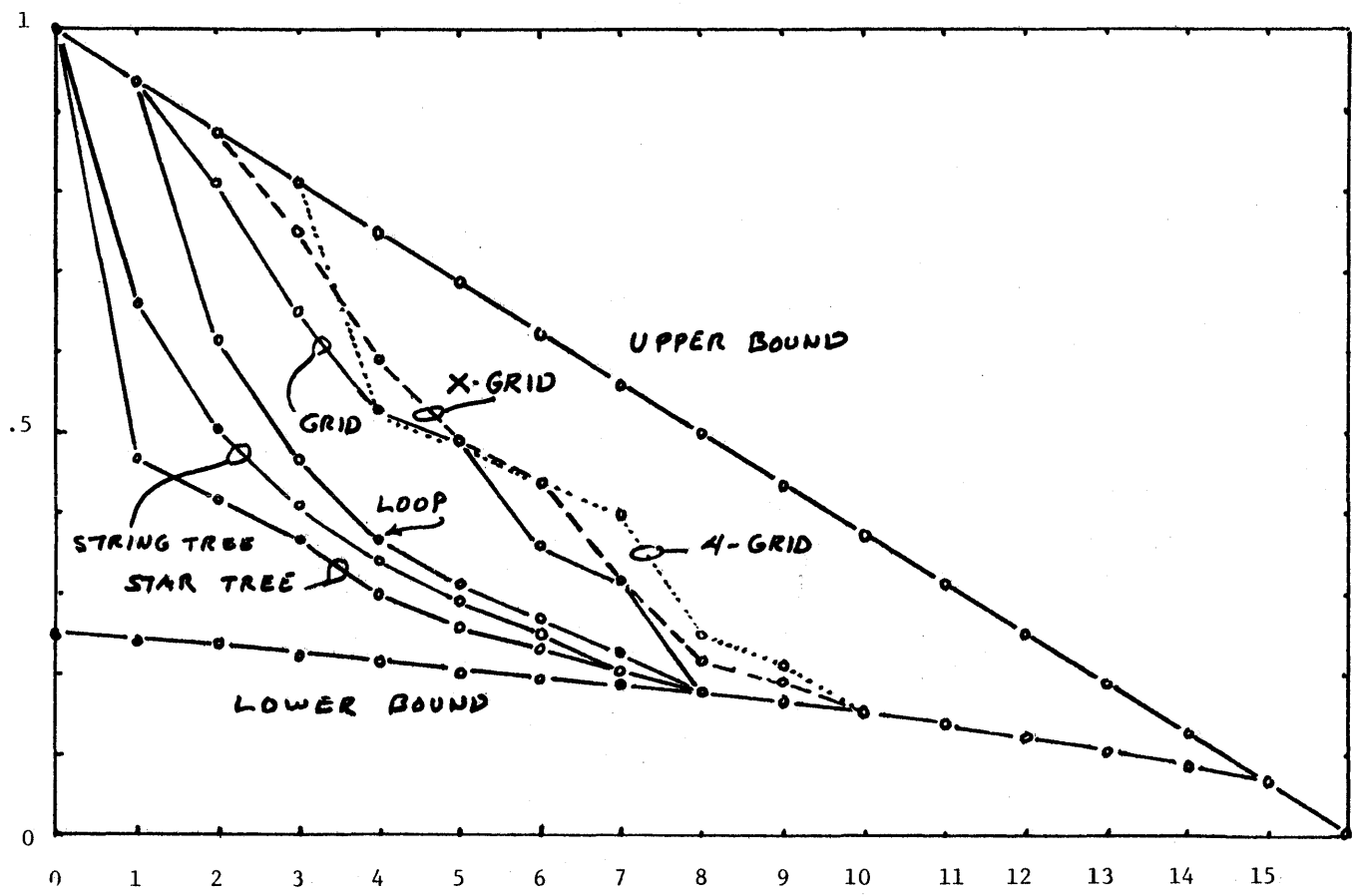


Figure 5—Node survival functions for 16-node examples.

neighbors. We assume further that the basis of comparison is ability to function under optimal node attack.

Figure 4 illustrates seven connection topologies in order of increasing cost. The connectivity parameter ( $\omega$ ) is given for each topology. Figure 5 is a plot of the node survival function ( $\mu_k$ ) for these topologies using the underlying survival index of

$$\sigma(x) = [\sum (x_i/n)^2]^{1/2}.$$

(As previously noted, this particular choice of  $\sigma$  creates a numerical scale, but any other choice of a survival index would not affect the vertical ranking of points on the graph.)

We can make several useful qualitative observations from study of this example:

- Topologies of equal connectivity (or cost) are not otherwise of equal strength (string tree vs. star tree, and loop vs. grid).
- Performance is not uniformly ordered by connectivity or cost ( $X$ -grid vs. 4-grid).
- Large cost increases may be required to get small improvements in overall survivability performance.

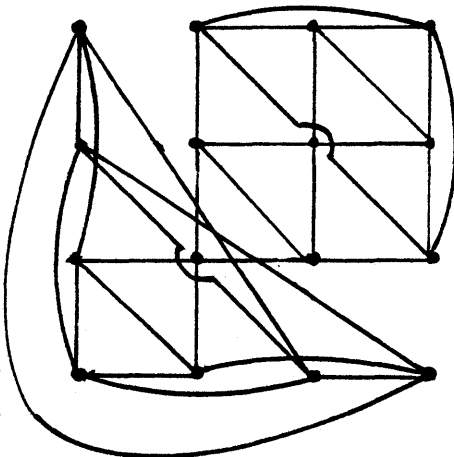


Figure 6—Degree-4 graph with single vulnerable node.

A final example is illustrative of the relationship between node degree and survival performance, and shows that even the connectivity is not determined by the minimum degree of the nodes. We know

$$\omega \leq \lambda \leq d_{\min},$$

and regular graphs can be constructed to make  $\omega = d_{\min}$ . But the inequality can exist. For example (see Figure 6), it is possible to make the 16-node graph regular (equal degree of all nodes) and of degree 4 but have  $\omega = 1$  (a single vulnerable node). Cost for this poor design is about 52 units.

## CONCLUSIONS

This paper has explored a new methodology for evaluating survival performance of distributed processing networks which are subject to losses of nodes and/or links due to failure or attack. The axiomatic definition of "survival index" accommodates a wide class of actual system performance characteristics, yet preserves rank. The present work while providing a rigorous mathematical structure, is restricted to the homogeneous case, and ignores other realities of real networks such as limited link bandwidth effects and control problems.

The basic advance over the prior notions of connectivity and cohesiveness are that performance degradation beyond the connected/disconnected point are measured.

The ranking equivalence of all survival indices satisfying the axioms provides a convenience in computer searches for optimal attacks or design evaluations and allows a freedom from concern with the mean of exact quantitative values in the model.

## REFERENCES

- Wilkor, R. S., "Analysis and Design of Reliable Computer Networks," *IEEE Transactions on Communications*, June 1972.
- Frank, H. and Frisch, I. T., *Communication, Transmission, and Transportation Networks*, Reading, Maryland: Addison-Wesley, 1971.





# Architectures for supersystems of the '80s

by SVETLANA P. KARTASHEV\*

*University of Nebraska-Lincoln*

and

STEVEN I. KARTASHEV

*Dynamic Computer Architecture, Inc.*

## 1. INTRODUCTION

Several areas of science and technology offer a number of problems which solutions require an enormous computational power. For instance, partial differential equations in computational aerodynamics, real-time radar signal processing, control of fast and complex nuclear or chemical reactions, accurate weather prediction, etc., necessitate throughputs which are several orders of magnitude higher than those of existing complex parallel systems.

The systems which possess an extreme computational power and are capable of solving such problems will be called *Supersystems*. The problems for Supersystems may be non-real-time and real-time.

A typical non-real-time problem is exemplified by computer simulation of fluid flow required in testing and validation of aerodynamic design process. A realistic aerodynamic flow requires an extremely high resolution of grid points appropriately spaced throughout the flow field. Further, it is necessary to simulate not only a two but a three dimensional flow in order to have complete and accurate results of simulation. Although such a powerful system as Illiac-IV was proven to be effective in simulation of two dimensional flows, a complex three dimensional simulation requires a system with nearly a two order of magnitude increase in throughput as compared to that of Illiac-IV [1].

As for complex real-time problems, one can refer here to Ballistic Missile Defense Algorithms which are characterized by thousands of information streams that have to be processed in real-time during minimal time intervals [2-5]. This means that these algorithms require that the Supersystem compute its responses during limited time intervals. Namely, there exists a time restriction between the moment of time a particular set of data streams enters the Supersystem and the moment of time the Supersystem computes a response specified by this set. For instance, for a Ballistic Missile Defense Algorithm computing a response for a sensor (radar) there exists a time constraint associated with computation of a sequence of commands for sensor which allows an im-

provement in observation of an object entering the field of view of the radar [3].

Therefore, complex real-time problems impose more severe requirements to Supersystems. Indeed, in addition to the demand to handle an enormous amount of information, they require that this information be handled during specified and, as a rule, very small time intervals. Thus real-time problems require Supersystems with the highest throughputs attainable by modern technological developments.

However, obtaining maximal throughputs is not the only goal in designing such systems. Another objective of no less importance is to have highly reliable computations. This acquires special significance for many complex real-time problems. Indeed, an error in computation of an algorithm which controls nuclear reactor may lead to catastrophic circumstances. Likewise, Ballistic Missile Defense Algorithms also necessitate a very high degree of computational reliability due to the criticality of the mission.

Therefore, designers of Supersystems are confronted with the following requirements: on one hand they are required to design systems of enormous complexity while, on the other, the systems designed must provide a specified level of reliability. These two requirements are contradictory because the system's reliability has a tendency of going down as its complexity goes up [6].

## 2. TRADITIONAL SOURCES OF AUGMENTING THE SYSTEM THROUGHPUT

One may increase throughput of a complex parallel system by drawing on the following traditional sources:

- I. faster hardware;
- II. modular expansion of the system with new equipment (adding of new computers, processors, I/O units, etc.);
- III. equipping the system with a dedicated architecture that is very effective for a given class of applications;
- IV. utilization of the maximal concurrency present in programs;
- V. application of special computations: pipeline, array, associative;

\* The research done by this author has been supported under Contract DAAG29-76-D-0100 from Battelle Columbus Laboratories under Scientific Services Program.

## VI. optimization in data exchanges between separate computers (processors).

Consider now the effectiveness of these sources in increasing throughputs of Supersystems.

I. *Faster hardware* means implementation of a Supersystem from the fastest components which are available at the moment the system construction begins.

Application of this source of augmenting a system's throughput is delimited by the moment the system development is over. Surely certain system's units (for instance, I/O terminals) may be replaced with faster ones during systems' exploitation. However, massive on-line replacement of all system units with newer and faster counterparts appears to be highly unlikely.

II. *Modular expansion* means integration of new resource units into an existing system. Presently, this source of throughput increase acquires a greater and greater importance since the hardware resource has a tendency to become cheaper.

However, since each attachment of additional hardware resource is accompanied by adding new interconnections, the modular expansion of any existing system is limited by one or a combination of the following factors:

a. The significant initial complexity of the Supersystem coupled with the serious requirement for maintaining a given level of its reliability restricts the amount of new hardware which can safely be added to the system. In addition, any modular expansion must be accompanied by the ability of the system's diagnostical network to maintain the initial diagnosability of the system. Otherwise, some faulty modules remain undetected and the system's reliability erodes. However, the initial diagnosability has a tendency to decline when a new equipment is added [7]. Thus preservation of initial reliability poses a limit on the number of hardware units which can be added to the system.

b. Current technological constraints may limit the number of new hardware components which can be attached to an existing system. For instance, a modern restriction on pin count per LSI module may restrict the overall number of new modules which may be attached to a system already constructed.

c. Expansion of the existing system with new resources may lead either to a significant increase in delays introduced by interconnection logic and/or to a partial or complete disruption in flexibility of inter connections among system's modules. It then follows that for any architecture there exists such a limiting complexity of its hardware resource that all throughput increases due to integration of new equipment into a constructed system may be offset by the time losses such integration incurs. This means that the system which achieves this complexity no longer increases its throughput. Thus a system achieves a boundary on its throughput which cannot be exceeded by any modular expansion of the resource.

III. *A dedicated architecture* is the architecture which takes into account computational specificities of an algorithm. Such architecture is provided with special purpose executional and control hardware. A special purpose executional hardware is understood as a set of special purpose

micro-operations which facilitate execution of an algorithm. A special purpose control hardware is understood as a set of dedicated instructions each of which implements complex sequences of operations encountered in an executing algorithm.

The major drawback of the architectural dedication is that it lowers the system applicability. If a new complex problem appears or an old problem is modified, a Supersystem might be incapable of solving it during the necessary time interval.

IV. *Utilization of maximal concurrency present in programs.* This means finding the maximal number of instruction and data streams present in the complex algorithm and assigning a separate computer (processor) to each such stream.

The major drawback of this technique is that it may lead to an excessive complexity of a Supersystem and a low utilization of the equipment it incorporates. Indeed, the number of independent computers in a system is specified by the algorithm having the maximal number of parallel information streams. On the other hand, the bit size of each computer is specified by the maximal bit size of computed results which can be encountered in all algorithms computed by this computer. It then follows that in computation of other less complex algorithms requiring smaller number of instruction and data streams or smaller word sizes, a portion of the system resource becomes redundant. However, the amount of the free resource formed in the system may be insufficient to organize concurrent computation of other algorithms. Thus this resource becomes idle.

V. *Application of special computations: pipeline, array, associative.* Complex algorithms may have portions (tasks) requiring different types of computations—multicomputer, array, pipeline. For instance, one task is characterized by a large number of parallel instruction streams with little interaction. Thus this task may be computed by a multicomputer system having a number of computers that match the number of instruction streams. Suppose that the next task of the algorithm employs a great deal of data parallelism, i.e. each program instruction has to handle a data vector of large dimension. Thus this task requires an array system. The next task may require an increased speed in each instruction stream, which can be accomplished through pipelining, etc. To compute this algorithm in minimal time, it must be computed by three separate subsystems employing different types of architecture: multicomputer, array, and pipeline. This leads to excessive complexity of the resource since each of the systems is engaged in computation of one task only; during execution of the remaining tasks it is idle.

VI. *Optimization in data exchanges between separate computers (processors).* One of the ways of performance improvement is in reducing the time of external information exchanges between computers. This may be achieved if data transfers through I/O devices of two computers are replaced by direct access of one computer to the primary memory or the processor of another computer. Thus in order to minimize the number of I/O exchanges a system must be provided with a flexible processor-memory bus which maintains direct processor-memory, processor-processor, and memory-memory exchanges between different computers.

In order to organize connections between all processors,

all primary memories and I/O devices, one has to have a bus of enormous complexity. Thus it is hardly conceivable that all functional units of the Supersystem could be connected with such a bus. The complexity of the bus can be reduced if the hierarchy of interconnections is introduced: namely, one-staged connectedness between all functional units of each subsystem; a one-staged connectedness between different subsystems, etc. As a result, the time of communications between a pair of functional units belonging to two subsystems will be significant, which will reduce throughput of the Supersystem.

Most of the traditional sources of increasing a system throughput considered above have an essential drawback—their use leads either to excessive complexity of the hardware resources of the Supersystem or to a limit in its applicability. Therefore one must consider new sources of throughput increase that would be capable of augmenting a system throughput without a significant increase in its complexity or a restriction on its applicability.

### 3. NEW SOURCES OF AUGMENTING THE SYSTEM THROUGHPUT

Advances of LSI technology in the 70's made it possible to obtain LSI modules with high computational throughput. The use of such modules as the building blocks of a Supersystem opens new options of increasing its power. Indeed, LSI technology has introduced modularity as basic in the organization of a computer's architecture. Each LSI module from which it is assembled may be equipped with simple circuits for the software-controlled activation and deactivation of interconnections with other modules. For instance, processor modules may be switched among several main memory modules, and this may reduce the time required for processor-memory communication. Or, one processor may reconfigure into several smaller-sized processors in an array parallel system, leading to an augmentation of the data vector size processed by a single instruction. Finally, a modular architecture supports a complete dynamic redistribution of available hardware by reconfiguring hardware resources into different numbers of variably sized computers. This allows a computer architecture to adjust to the changeable number of information streams encountered in complex algorithms.

It then follows that a computer architecture assembled from LSI modules may exhibit a much higher adaptation to executing algorithms than was ever achieved in traditional systems. Let us consider these new sources of adaptation and the way they are applicable to Supersystems.

#### 3.1. Adaptation of hardware resources on instruction and data parallelism

During the past few years there appeared a number of publications [3,8-12] describing so called dynamic architectures of parallel computer systems. These architectures are capable of dynamically partitioning their hardware resources into a variable number of computers (processors). This allows a system accommodation to a current number of in-

struction and data streams it needs to handle. Accordingly, if the system has to process an increased number of information streams, it may partition its resources into a larger number of computers with smaller word sizes. Consequently, for a complex real-time algorithm, a maximal number of its information streams can be handled in real time, although precision of computations may reduce. In case a particular information stream requires a more precise computation, a larger size computer (processor) may be formed for its handling so the accuracy of computations will be enhanced.

Note: the capability of dynamic architectures to perform dynamic partitioning of the resource allows computation of each information stream in a computer with the minimal word size. This leads to maximization in the number of information streams computed by the available resources. Therefore established is a new source of throughput increase through dynamic adaptation to the current instruction and data parallelism present in a complex algorithm. This source was absent in traditional systems. Its advantage is that it is not accompanied by significant increases in system complexity, since it is accomplished through redistribution of the available resources.

*Example 1.* Let a Supersystem include one hundred 64-bit computers, eighty 48-bit computers and fifty 32-bit computers, i.e., the system resource,  $SR = [100 \times 64 \text{ bits}, 80 \times 48 \text{ bits}, 50 \times 32 \text{ bits}]$ . Suppose that this Supersystem has to compute three complex algorithms with the following resource requirements: The resource needs for Algorithm I,  $RA_I = [150 \times 48 \text{ bits}, 25 \times 32 \text{ bits}]$ , i.e., it needs one hundred fifty 48-bit computers and twenty-five 32-bit computers; the resource needs for Algorithm II,  $RA_{II} = [200 \times 32 \text{ bits}]$ , and for Algorithm III,  $RA_{III} = [40 \times 64 \text{ bits}, 25 \times 32 \text{ bits}]$ . If the Supersystem is equipped with a conventional architecture, i.e., it is incapable of redistributing its resources, the redundant resources,  $RR$ , it has in execution of Algorithms I, II, III, respectively, are shown in Table I.

The Supersystem may execute Algorithm I in 80 48-bit computers, 70 64-bit computers and 25 32-bit computers. Its redundant resources that can be used for other computations are  $30 \times 64 \text{ bits}, 25 \times 32 \text{ bits}$ . In addition, in each 64-bit computer that computes Algorithm I its 16-bit portion is redundant. However, the resulting resource ( $70 \times 16 \text{ bits}$ ) cannot be released for other computations. In computation of Algorithm II, the following system resource is engaged in computations:  $50 \times 32 \text{ bits}, 80 \times 48 \text{ bits}, 70 \times 64 \text{ bits}$ . Thus, the system redundant resource that can be freed for other computations is  $30 \times 64 \text{ bits}$ . The redundant resource that cannot be freed is  $80 \times 16 \text{ bits}, 70 \times 32 \text{ bits}$ . Similarly, one can find that for Algorithm III, the system has free redundant resource  $60 \times 64 \text{ bits}, 80 \times 48 \text{ bits}, 25 \times 32 \text{ bits}$  and no non-free redundant resource.

It follows, that if the Supersystem employs a conventional architecture, it does not have enough resources to compute two algorithms concurrently.

Let us show that if the same Supersystem is equipped with dynamic architecture then it has enough resources to compute concurrently two algorithms (Table II). Indeed, a dynamic architecture may switch its resource into a set of architectural states distinguished from each other by the

TABLE I.—Supersystem with Conventional Architecture

Computational Modes	Resource Requirements of Algorithms			Redundant Resource (RR)	
	Algorithm I (RA <sub>I</sub> )	Algorithm II (RA <sub>II</sub> )	Algorithm III (RA <sub>III</sub> )	Free Resource	Non-Free Resource
Computation of Algorithm I	150 x 48 bits 25 x 32 bits	---	---	30 x 64 bits 25 x 32 bits	70 x 16 bits
Computation of Algorithm II	---	200 x 32 bits	---	30 x 64 bits	70 x 32 bits 80 x 16 bits
Computation of Algorithm III	---	---	40 x 64 bits 25 x 32 bits	60 x 64 bits 80 x 48 bits 25 x 32 bits	

numbers and sizes of concurrently operating computers. Any computer is assembled from  $k$  computer elements, CE, having the same word size  $h$ . Thus it handles  $k \cdot h$ -bit words where  $k = 1, 2, \dots, n$ . [10] If one assumes that  $h = 16$  bits, then 32-bit computer is assembled from two CE, a 48-bit computer is assembled from three CE and a 64-bit computer is assembled from four CE. Then the entire Supersystem resource is equivalent to 740 CE, because  $100 \times 64$  bits = 400 CE,  $80 \times 48$  bits = 240 CE,  $50 \times 32$  bits = 100 CE and  $SR = 400 + 240 + 100 = 740$  CE. Find the number of CE's required for each algorithm. Algorithm I requires 500 CE ( $150 \times 48$  bits = 450 CE,  $25 \times 32$  bits = 50 CE). Algorithm II requires 400 CE ( $200 \times 32$  bits = 400 CE). Algorithm III requires 210 CE ( $40 \times 64$  bits = 160 CE,  $25 \times 32$  bits = 50 CE). Therefore, the Supersystem has enough resources to compute concurrently Algorithms I and III ( $500 + 210 = 710 < 740$ ) and Algorithms II and III ( $400 + 210 = 610 < 740$ ). Concurrent execution of Algorithms I and II is impossible since their re-

source needs are  $500 \text{ CE} + 400 \text{ CE} = 900 \text{ CE}$  and Supersystem has only 740 CE.

In execution of Algorithms I, III, the Supersystem assumes the architectural state [ $150 \times 48$  bits,  $50 \times 32$  bits,  $40 \times 64$  bits]. The redundant resource equivalent to  $30 \times 16$  bits can be freed to other computations. In execution of Algorithms II and III, it assumes the architectural state [ $225 \times 32$  bits,  $40 \times 64$  bits]. Its redundant resource equivalent to  $130 \times 16$  bits can also be used for other computations. Therefore the Supersystem with dynamic architecture allows an additional throughput increase using the same complexity of the resources. ■

### 3.2. Reconfiguration of hardware resources into different types of architecture

Let us consider one more source of increasing throughput in a Supersystem via reconfiguring available hardware re-

TABLE II.—Supersystem with Dynamic Architecture

System Resource = [100 x 64 bits, 80 x 48 bits, 50 x 32 bits]

Computational Modes	Resource Requirements of Algorithms			Redundant Resource (RR)
	Algorithm I (RA <sub>I</sub> )	Algorithm II (RA <sub>II</sub> )	Algorithm III (RA <sub>III</sub> )	
Concurrent Computation of Algorithms I and III	150 x 48 bits 25 x 32 bits	---	40 x 64 bits 25 x 32 bits	30 x 16 bits
Concurrent Computation of Algorithms II and III		200 x 32 bits	40 x 64 bits 25 x 32 bits	130 x 16 bits

sources into different types of architecture: multicomputer, multiprocessor, array, pipeline.

Many complex real-time algorithms may be partitioned into portions (tasks) requiring different types of computations. To speed up execution, they are now executed by dedicated subsystems: array, pipeline, etc. This adds excessive complexity to the system and lowers its throughput since each of the dedicated subsystems is engaged in execution of only one task with matching dedication. During execution of the remaining tasks, as a rule, it is idle.

To increase throughput of the Supersystem, it is necessary that all its resources be continuously involved in computations. This implies that the system must be capable of switching its resources into different types of architecture: array, pipeline, multicomputer, multiprocessor. Or it must have co-residence of any combination of these architectures, namely, a portion of the resource functions as a multiprocessor. Another one behaves as an array and/or pipeline, etc.

Let us consider how such reconfiguration of the architecture may speed up computations in Supersystem.

#### Multicomputer computations

A Supersystem should assume a multicomputer mode of operation in executing those complex real-time algorithms that are characterized by a large number of concurrent in-

struction streams with little interaction. To perform this adaptation, the system should reconfigure into a state of the multicomputer architecture specified by a required number of concurrent computers. This will allow one to implement all instruction parallelism present in algorithms because the system will follow all changes in instruction streams by switching into states characterized by matching number of computers.

As for precision of computations, this will depend on the priority of programs requesting a high precision computation. If the priority of this program is high, then the Supersystem should redistribute the resource assigned for other programs and form a computer size needed by a high priority program.

#### Multiprocessor computations

The Supersystem should assume a multiprocessor architecture while executing complex algorithms with high interaction among instruction streams, i.e., one program may require data words computed by any other program.

*Example 2.* Let program  $P_1$  computed in the A processor assembled from processor elements  $PE_1$  and  $PE_2$  need a data array obtained by program  $P_2$  which was computed before by the B processor made of  $PE_3$  and  $PE_4$  and stored in its local memory B (Figure 1). To perform this interaction the

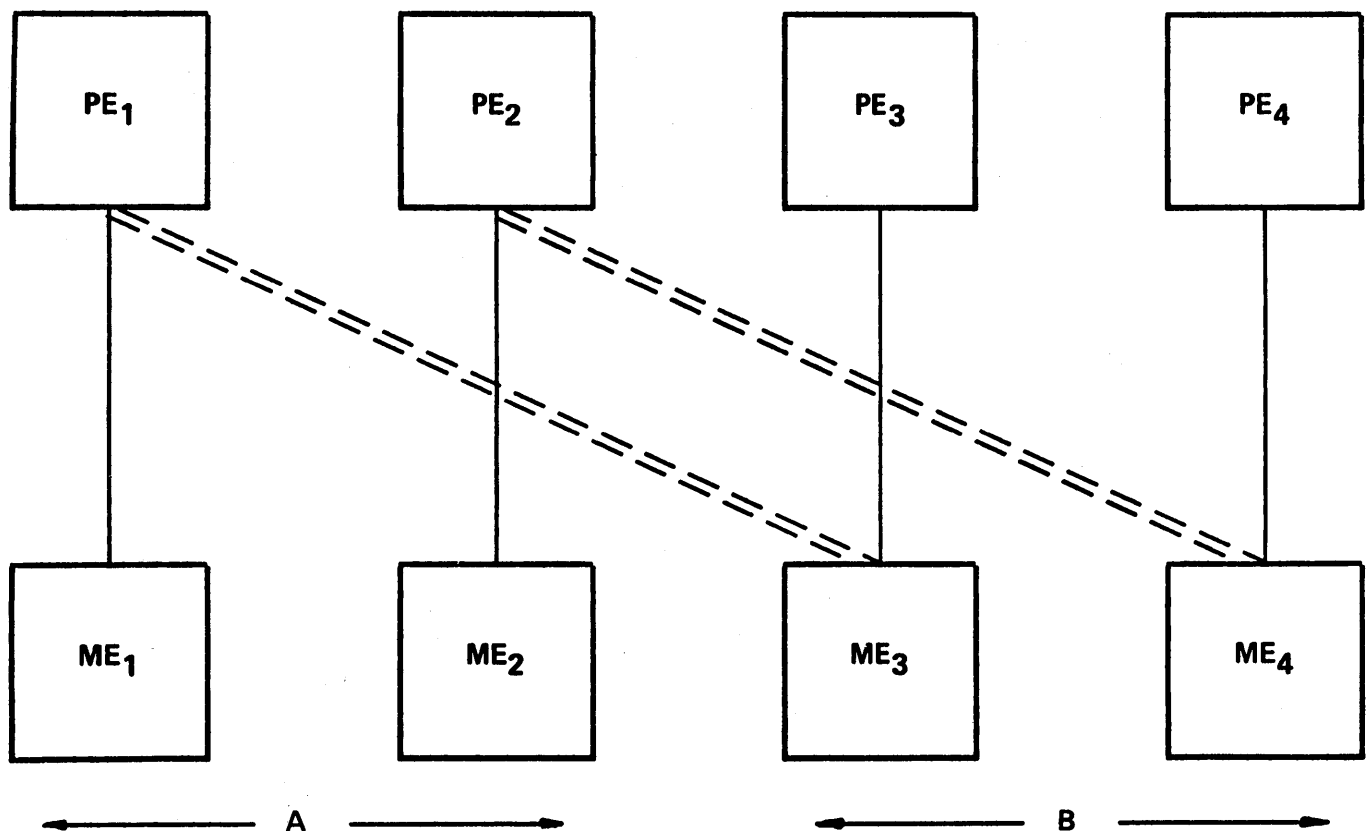


Figure 1—Multiprocessor exchange.

A processor should establish direct communication with the B memory. This will allow the A processor to compute in the mode when it fetches the first operand from its local A memory and the second operand from the B memory. Thus this mode will eliminate time overheads associated with transfers of blocks of data words from the memory of one processor to that of another, which is extremely important for complex real-time algorithms requiring very fast computations. ■

The same mode can be organized on the level of byte exchanges, when one processor fetches a portion of the word from the memory of another processor. Indeed, since the system may compute several programs handling data with different word sizes, a program handling data with smaller word size may need an array of data which matches its size and which is stored in the memory of a larger size processor. To perform this interaction, the system assumes byte exchange between the processor and the memory of two different computers.

*Example 3.* Let a 16-bit processor C assembled from PE<sub>4</sub> need 16-bit addresses computed before by 32-bit processor A assembled from PE<sub>1</sub> and PE<sub>2</sub> and stored in its local memory element ME<sub>2</sub> of its primary memory assembled from ME<sub>1</sub> and ME<sub>2</sub>. For this case C processor establishes a direct communication with a 16-bit portion of the 32-bit primary memory of A processor and performs a parallel byte exchange (Figure 2). ■

**Array computations**

In executing complex real-time algorithms requiring array computations, a portion or the totality of the Supersystem should assume array architecture. In this case it will be able to change the number of concurrent arrays, the dimension of a data vector computed in each array, and the word sizes of processors working in each array. For each array, its resource should be redistributed among its processors, so that a data vector computed in each array may contain different size data items. Therefore, the architecture of the Supersystem should be able to perform array computations with variable precision and variable dimension of data vectors.

**Pipeline computations**

Pipelining is an attractive choice for very fast computations since it allows elimination of the time of memory accesses from the time of program execution. The major problem with modern pipelines is the time overheads introduced because of the disparity between the pipeline(s) and the algorithm being executed. As a result, pipeline systems tend to be dedicated to certain types of computations and usually have a limited applicability.

To broaden the range of their cost-effective application,

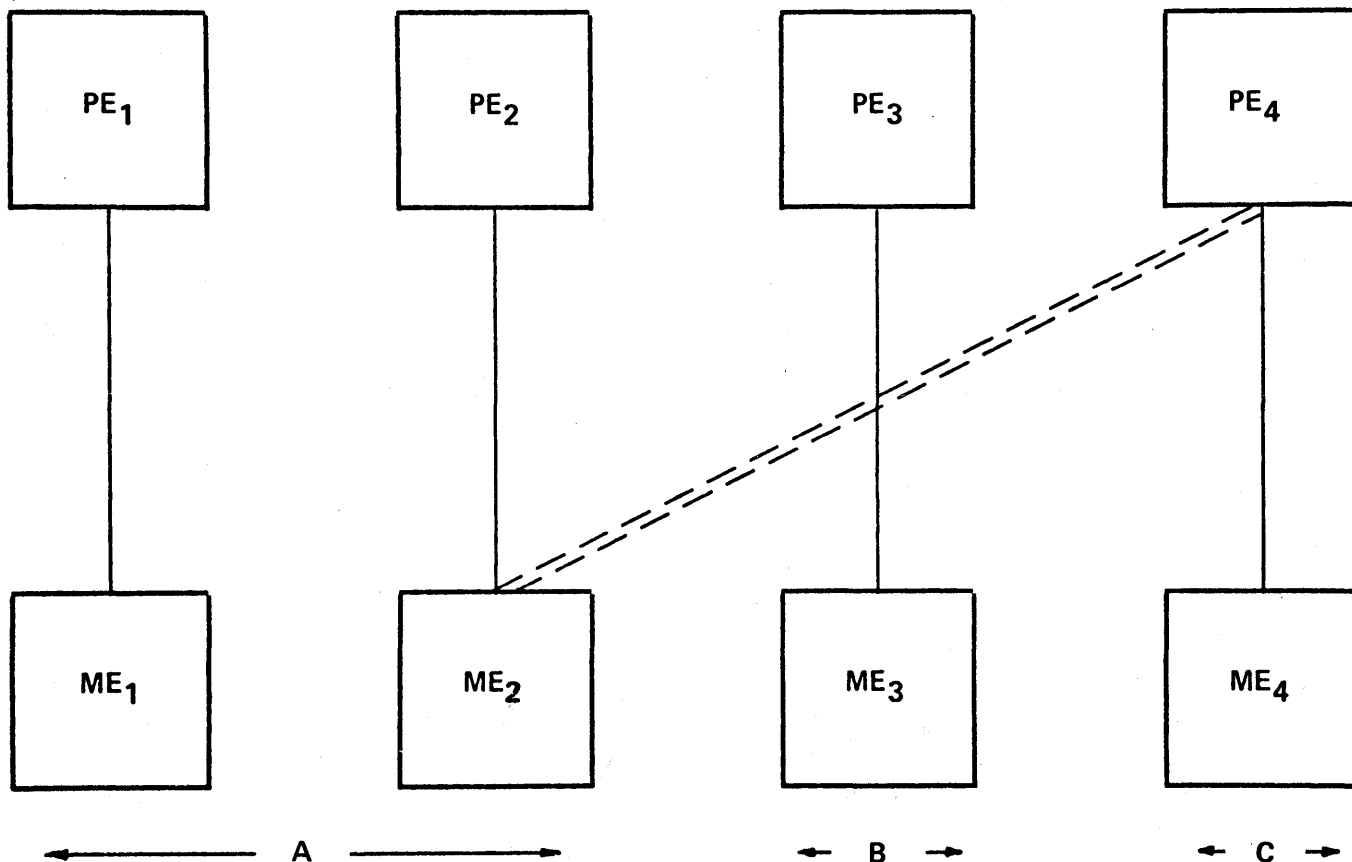


Figure 2—Multiprocessor byte exchange.

pipelines offer various software controllable reconfigurations of the available hardware resource. The general idea is to reconfigure the resource via software to reduce the dissimilarity between an arithmetic pipeline and sequences of operations assigned to program instructions [13-15]. However, since many complex algorithms may have different sequences of operations following each other, a pipeline that executes such an algorithm must reconfigure each time it switches from one sequence of operations to another. If  $\Delta t$  is the time required for each reconfiguration and  $N$  is the number of different operation sequences, then the pipeline loses time  $\Delta t \cdot N$  reconfiguring its resource. If  $\Delta t$  or  $N$  is large, then the speed advantage of pipelined computation may be lost and there is no sense in computing this algorithm in a pipeline. To reduce  $N$ , algorithms are sometimes rearranged into tasks where each task may be computed by a single configuration of the pipeline. But this requires special programming which may again restrict the class of programs that can cost-effectively be pipelined.

Therefore, a major thrust of an architectural research in pipelines has to be directed at broadening their applicability. Ideally, a pipeline system must be able to compute any program as cost-effectively as a general purpose computer. In [16,17] one pipeline system with dynamic architecture was described which performs a high degree of architectural adaptation toward executing algorithms. It eliminates the need for pipeline reconfiguration and its immediate consequences: time overheads and program restructuring. Programming for a dynamic pipeline is very simple, and practically it does not deviate from programming for conventional computers. This dynamic pipeline may be conceived as a new type of a general purpose computer that may perform pipelined computations. It allows effective organization of computations bearing no penalty for dedication which is the price all existing pipelined systems pay for the performance gains they achieve.

Inasmuch as a dynamic pipeline is assembled from the same building blocks that are used for multicomputer, multiprocessor and array systems with dynamic architecture [8,10,11], one may use these techniques in order to reconfigure a portion of the resources of Supersystem into one or several dynamic pipelines to be used for algorithms requiring pipeline computations.

### Mixed computations

A Supersystem may need concurrent co-residence of several types of architecture while executing a set of complex real-time algorithms requiring different computations. Or concurrent tasks within the same algorithm may need different architectures in order to be executed during minimal times. Thus the Supersystem should assume a mixed architecture mode when portions of its resources reconfigure into different types of architectures. This feature will allow performing of concurrent multicomputer, multiprocessor, array and pipeline computations using the same hardware resources instead of implementation of separate dedicated subsystems.

## 4. ARCHITECTURAL RECONFIGURATION IN SUPERSYSTEM

As follows from the above, new sources of throughput increases in a Supersystem can be realized via reconfiguration of the available resources. This reconfiguration is required in both cases: either to achieve necessary instruction or data parallelism or to adapt to a current type(s) of computations. Since during each reconfiguration the affected resources stop computation, reconfiguration introduces the time overheads into the time of computation. This implies that in order to make effective both sources of throughput increases the time of reconfiguration should be minimized. Then the Supersystem will be capable of performing very fast dynamic adaptations to complex algorithms it computes introducing minimal time losses in rearranging its resources into new architectures.

Let us consider what factors affect architectural reconfiguration in a system. As was shown in [11], to perform reconfiguration the following actions have to be performed:

First, new control codes have to be written to all modules of the resource requested for reconfiguration.

Second, new transfer modes have to be established in connecting elements of all reconfigurable buses destined to integrate requested modules into new computers, arrays, pipelines or separate them from each other.

### 4.1. Allocation of the array of control codes

Being written to the resource modules, control codes affect activation of new pattern of interconnections for the processor signals (carry, overflows, equality), they allow generation of new time intervals by the control units of all newly formed computers, arrays and pipelines, they distinguish different types of architecture to be assumed, etc.

Each architecture is characterized by a set of control codes which must be written to the modules of requested resource. Since a Supersystem may assume  $N$  different architectures, it is specified by  $N$  different sets of control codes. Thus all control codes form an array which dimension depends on how many architectures may be assumed by the Supersystem.

*Example 4.* Consider now how an array of control codes can be specified for a Supersystem with dynamic architecture assuming a multicomputer mode of operation, i.e., the entire resource may be redistributed among several programs to satisfy a present instruction parallelism. As was shown in [11], to form a new  $16 \cdot k$ -bit computer, one has to write four control codes to all its computer elements CE's:  $k$  (computer size code),  $p$  (processor code),  $b$  (significance code), and  $m_E$  (data fetch code). Codes  $k$  and  $b$  participate in forming a new end-around carry path for a newly formed computer, whereas codes  $p$  and  $m_E$  allow this computer to generate new time intervals for its processor dependent and memory access operations.

Storage of all variable control codes for one CE takes one 16-bit cell. Since a multicomputer dynamic architecture has  $n$  CE's, the control codes for an architectural state may be stored in a single  $16n$ -bit cell composed of " $n$ " 16-bit mem-



ory bytes where each byte stores control codes for one CE. Because this multicomputer architecture may assume  $2^{n-1}$  states [10], to store the control codes for all architectural states requires an array of  $2^{n-1}$   $16 \cdot n$ -bit cells. This array may be mapped either into I/O memory or primary memory. If it is stored in the initial cells of I/O memory, then the effective address of one  $16 \cdot n$ -bit cell may be made the code of the architectural state,  $N_f$ . Namely,  $N_f$  is the address of the  $16 \cdot n$ -bit cell which stores all the necessary control codes which have to be written to requested CE when  $N_f$  state is established. This means that when  $N_f$  cell is accessed, only its memory bytes corresponding to requested CE's have to be accessed. ■

It follows from the above that, in order to minimize the time of reconfiguration, such storage of the array of control codes has to be organized, that minimizes the time of accessing a set of control codes which distinguishes a given architectural state. In addition, being fetched, each set of control codes has to be written only to modules of requested resource. This means that if some modules are not requested for reconfiguration they have to receive no new control codes, because they are integrated into the same computer, array, pipeline which continues to function in both architectural states. One of the techniques of storing an array of control codes was presented in [8]. The time to access any item in this array does not exceed the time of 64-bit addition.

#### 4.2. Switching reconfigurable buses

Dynamic architecture may have two types of reconfigurable buses:

- (1) Processor (I/O) Bus that interconnects processor modules of the resource and allows formation of new paths for the following processor signals: carry, overflows, equality. The need for this bus is caused by variation in computer or processor sizes which is performed in dynamic architecture.
- (2) Memory Processor Bus that interconnects all processor and memory resources and provides separation of instruction and data streams computed by concurrent computers, arrays and pipelines.

Appearance of the two dedicated buses is caused by the fact that a Processor Bus introduces a smaller reconfiguration delay than a Memory Processor Bus [17]. Thus it may be made much faster. Otherwise if both buses share interconnections, then the time of clock period will depend on the time of signal propagation in the Memory Processor Bus. This will lead to a significant slowdown in the rate of information processing. As was shown in [11] to form a new end-around carry path in the Processor Bus, it is sufficient to write only several control codes in the processor modules of requested resource. Thus a fast processor bus may be organized without connecting elements. The time of reconfiguration of such bus will be entirely dependent on the time of accessing new control codes.

As for the Memory Processor Bus, this bus necessarily includes connecting elements which modes of transfer may

be switched during reconfiguration. This implies that the time of reconfiguration for the Memory Processor Bus consists of the time of switching connecting elements into such transfer modes which are determined by a new architectural state.

The Memory Processor Bus must possess several important attributes. If the resource is formed into several independent computers, pipelines or arrays, then the bus must separate their instruction and data streams. Within each computer, pipeline and array the bus must provide the instruction fetch from the memory module which stores a currently executed program segment to all other modules they integrate. Also it must maintain parallel information exchange with variable size words between the primary memory and the processor of each computer, pipeline and array.

These are the two essential requirements toward Memory Processor Bus without which a dynamic architecture will not function at all. However, significant performance gains are achieved if the Memory Processor Bus satisfies an additional flexibility requirement, which provides that for any two processors in the resource the bus must implement a direct access of one processor to the entire primary memory or any combination of primary memory modules integrated by another processor; parallel information exchanges between these processors or their separate modules; parallel information exchanges between primary memories or separate modules of these memories. In addition, the bus must allow the maximal number of communicating pairs (processor-processor, memory-memory, processor-memory) be involved in communications.

It follows from the above that performance optimization for the Memory Processor Bus includes two factors:

- (1) minimizing the time of reconfiguring this bus into a new architectural state; and
- (2) for each architectural state, the bus has to minimize the time of information exchanges between any pair of resource modules involved in communication.

## 5. UNIVERSAL DC GROUP AS A NEW BUILDING BLOCK OF SUPERSYSTEM

Above it was shown that an adaptation of hardware resources on instruction and data parallelism and their reconfiguration into different types of architecture are new important sources which may lead to a significant increase in a throughput of the Supersystem. Let us introduce one building block called a Universal Dynamic Computer Group (UDC group) that may realize these sources of throughput increases.

### 5.1. General description of the hardware resources

The UDC group is assembled from  $n$  computer elements CE, reconfigurable Memory Processor Bus, and V-monitor equipped with the memory  $M(V)$  [18]. Each CE processes 16-bit words in parallel and includes one processor element, PE, one memory element, ME, and one I/O element, GE,

having the same word size (16 bits). Each GE has small memory  $M(GE)$  with the same word width. In Figure 3, the UDC group assembled from four CE is shown. The system will form variable size computers (processors)  $C_i(k)$  containing  $k$  CE where  $i$  is the position of the most significant CE. In  $C_i(k)$ , the integer  $k$  may range from 1 to  $n$ ; accordingly, each  $C_i(k)$  may have a word size changing from 16 to  $16n$ -bits in 16 bit increments. Thus a dynamic architecture may partition the hardware resource into the following word sizes (in bits) of concurrent computers: 16, 32, 48, ...,  $16n$ -bits.

The reconfigurable Memory Processor Bus contains two types of connecting modules: address connecting element (ASE) and memory connecting element (MSE). The ASE elements connect each processor element (PE) with the memory elements (ME) and broadcast memory address and read and write signals from this PE to ME. The number of ASE elements assigned to each PE is  $n$  where  $n$  is the number of ME's in the resource, so that  $ASE_i$  transfers address and two signals from PE to  $ME_i$  ( $i=1, \dots, n$ ). The MSE elements are connected to each ME and exchange  $h$ -bit bytes between this ME and PE's. Each ME is assigned with  $n$  MSE, where

$n$  is the number of PE's in the resource so that  $MSE_i$  exchanges  $h$ -bit bytes between this ME and PE $_i$ .

Consider how one may establish a communication path between a pair of PE and ME elements. To activate information broadcast between a pair of PE and ME, one has to establish two separate paths: an address path whereby PE broadcasts an address to ME, and data path whereby one element PE or ME sends a data byte to another element of the pair.

Any processor element PE may generate address and read and write signals for any memory element ME in the resource. PE outputs this address through the Address channel (A-channel) connected with all ASE (Figure 4). Selection of the ASE that broadcasts this address and read or write signals is specified with the Selective Activation code (SEL) sent through channel B. SEL is  $\log_2 n$  bit code where  $n$  is the number of ASE connecting elements assigned to each PE. Its meaning equals binary meaning of position  $i$  of  $ASE_i$ , which has to broadcast the address ( $SEL=i$ ). Each ASE stores its own position code  $i$ , so that  $ASE_1$  stores  $i=1$ ,  $ASE_2$  stores  $i=2$ , etc.

Upon receiving code SEL, each ASE compares it with its

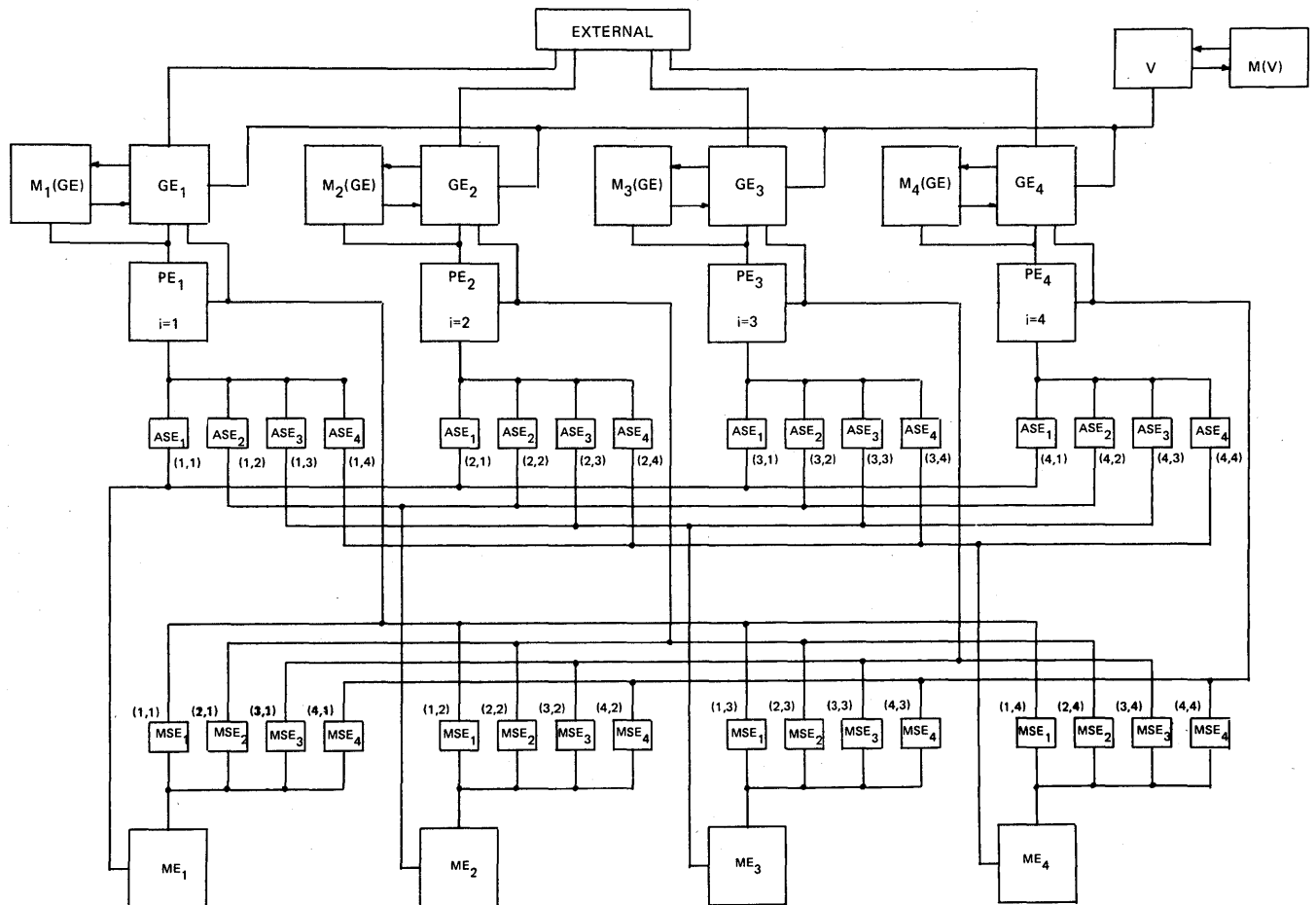


Figure 3—Hardware diagram for the ODC group.

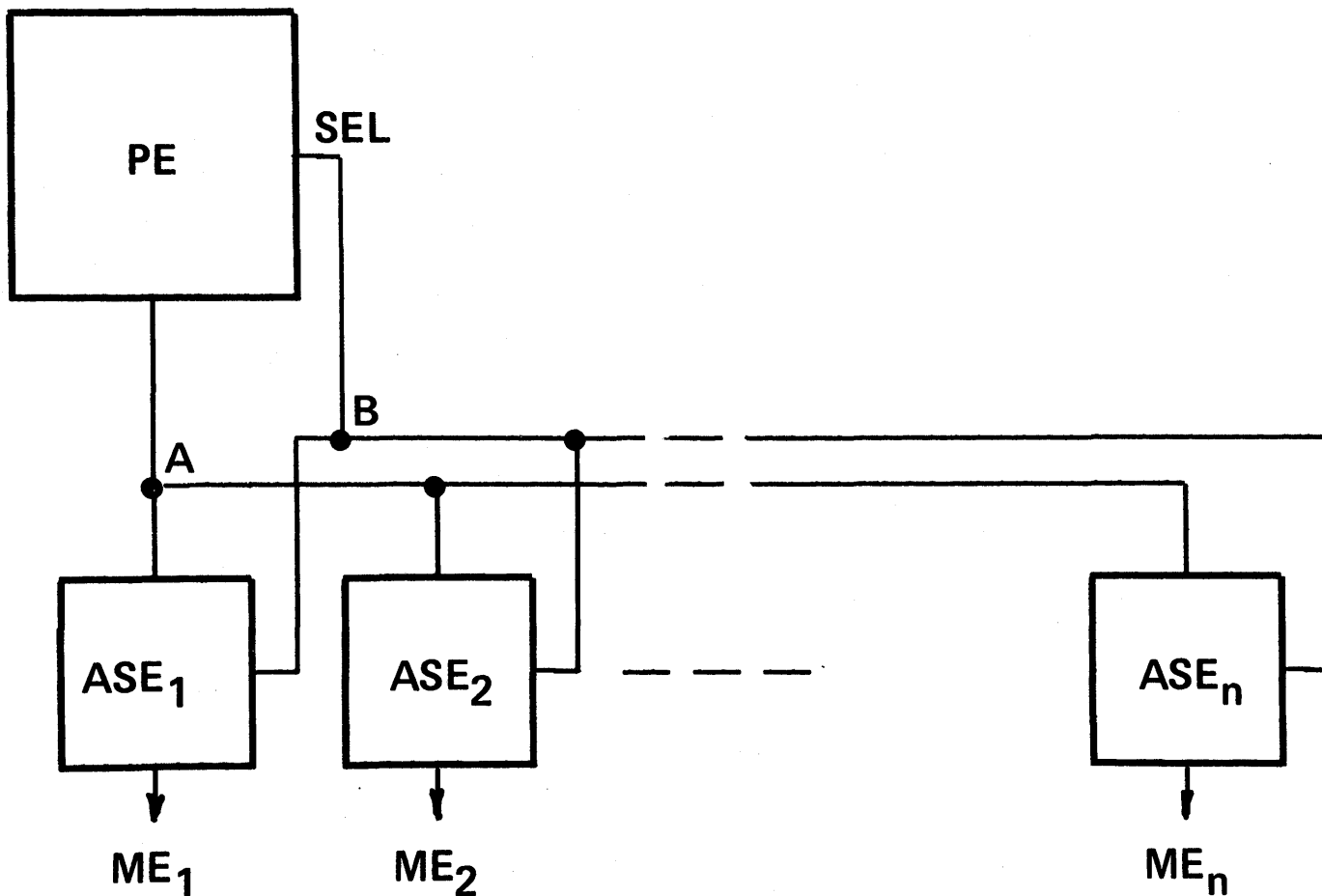


Figure 4—Organization of the address broadcasts for the ODC group.

own position code  $i$ . The ASE in which  $SEL = i$  is opened for broadcasting address to  $ME_i$ . For example, let  $PE_3$  must send address to  $ME_1$ . It sends  $SEL = 1$  through the B channel which activates  $ASE_1$ . Thus  $ASE_1$  receives address through the A channel and broadcasts it to  $ME_1$ .

The 16-bit byte exchange between a pair of PE and ME is accomplished through the connecting element MSE. Consider how this MSE is selected. For each ME assigned with  $n$  MSE, selective activation of  $MSE_j$  leads to establishing  $h$ -bit data path between this ME and  $PE_j$  (Figure 3). Selective activation of  $MSE_j$  is performed with the read signal ( $w_1$ ) if 16-bit byte has to be read from ME or the write signal ( $w_2$ ) if 16-bit byte has to be written to ME. These two signals are obtained from the connecting element ASE which broadcasts address and  $w$ 's to this ME. It then follows that to implement such selective activation one has to connect each connecting element  $ASE_j$  (which broadcasts address for  $ME_j$ ) and each connecting element  $MSE_i$  (which broadcasts  $h$ -bit byte between  $PE_i$  and  $ME_j$ ) with a 2-line connection  $(i,j)$ , where one line is used for  $w_1$  signal and another one is used for  $w_2$  signal.

Therefore, to organize 16-bit data path between  $PE_i$  and  $ME_j$  one has to construct the  $(i,j)$  connection using the following rule:  $PE_i$  broadcasts address through its  $ASE_j$  ( $i \rightarrow j$ ),

likewise  $MSE_i$  assigned to  $ME_j$  broadcasts the respective  $h$ -bit byte to or from  $ME_j$  ( $i \rightarrow j$ ). It then follows that the  $(i,j)$  connection is the connection for  $w_1$  and  $w_2$  signals between  $ASE_j$  (assigned to  $PE_i$ ) and  $MSE_i$  (assigned to  $ME_j$ ). In each ASE and MSE the  $(i,j)$  connection takes only 2 pins. Therefore, in order to implement the address and  $h$ -bit data paths described above the reconfigurable Memory Processor Bus has to have all  $(i,j)$  connections, where  $i, j = 1, \dots, n$ .

*Example 5.* Let processor element  $PE_2$  write 16-bit byte into a cell of memory element  $ME_4$  (Figure 3). This processor element sends the address of this cell and  $w_2$  signal through its A-channel (Figure 4) and code  $SEL = 4$  through its B-channel, thus selecting  $ASE_4$ . The  $ASE_4$  broadcasts address and  $w_2$  to  $ME_4$ . Since the  $(2,4)$  connection connects  $ASE_4$  assigned to  $PE_2$  with  $MSE_2$  assigned to  $ME_4$ ,  $ASE_4$  also sends  $w_2$  signal to  $MSE_2$  activating it into a mode of  $h$ -bit byte transfer to  $ME_4$ . Therefore  $PE_2$  broadcasts  $h$ -bit byte to  $ME_4$  through the  $MSE_2$  assigned to  $ME_4$ . ■

## 5.2. Multicomputer and multiprocessor architectures

In the UDC group multicomputer and multiprocessor architectures merge. Indeed, since any functional module of

UDC group (PE, GE and ME) may be directly connected with any other functional module without the use of I/O devices, the UDC group may use the flexibility of interconnections provided by reconfigurable multiprocessor architecture. On the other hand, the UDC group may form variable size computers from the available resources, thus taking advantage of the multicomputer architecture. Namely, for a  $16 \cdot k$ -bit computer assembled from  $k$  computer elements, CE, it is shown in [17] how the reconfigurable bus may establish (a) the instruction path, whereby an instruction stored in one CE may be transferred to all other CE's of the computer, and (b) parallel data path whereby the processor and memory of  $16 \cdot k$ -bit computer may have parallel exchanges with  $16 \cdot k$ -bit words. Therefore, the UDC group is capable of taking advantage of both multicomputer and multiprocessor architectures.

Let us show how the multiprocessor architecture of the UDC group allows a direct communication between different functional units of two computers which can be formed in a single architectural state.

For any pair of concurrent computers, A, B, the reconfigurable Memory Processor Bus allows a parallel communication between their processors, primary memories, and the processor of one computer and the primary memory of another. Consider first the communication between two processors of A and B computers.

### 5.2.1. Parallel exchange between the processors of two computers

To perform such an exchange each processor element, PE, of the A computer has to be connected in parallel with a matching PE of the B computer. The Memory Processor Bus may establish each such connection between two PE's in parallel through the path made of a pair of two connecting elements MSE (assigned to one memory element ME) which may pass a 16-bit byte in opposite directions (Figure 3).

Indeed, for each memory element ME, its  $n$  connecting elements, MSE, connect this ME with  $n$  processor elements, PE. Therefore to establish the 16-bit path between  $PE_a$  and  $PE_d$  one may use two MSE connecting elements  $MSE_a$  and  $MSE_d$  assigned to one ME, that connect this ME with  $PE_a$  and  $PE_d$  respectively. In addition, it is necessary that  $MSE_a$  be activated by the write signal  $w_2$ ; then it will pass the 16-bit byte from  $PE_a$  to the memory input, whereas  $MSE_d$  has to be activated by the read signal,  $w_1$ ; then it will pass the 16-bit byte from the memory input to  $PE_d$ .

*Example 6.* Let the resource form two computers A and B, which processors have to be engaged in direct communications. Namely, the A processor ( $PE_1, PE_2$ ) has to receive 32-bit words from the B processor ( $PE_3, PE_4$ ), since it computes second operands needed by the A computer. In order to establish a parallel connection between the two processors, the A computer executes a special instruction [17] that establishes a 32-bit data path between two pairs of communicating PE:  $PE_1, PE_3$  and  $PE_2, PE_4$ .  $PE_1$  communicates with  $PE_3$  through the path made of  $MSE_1$  and  $MSE_3$  belonging to  $ME_3$  (Figure 5).  $MSE_1$  is activated by the write

signal  $w_2$  and passes a 16-bit byte to the memory input;  $MSE_3$  is activated by the read signal,  $w_1$ , and passes this byte from the memory input to  $PE_3$ . A second communicating pair,  $PE_2, PE_4$ , establishes direct 16-bit data path through the pair of  $MSE_2$  and  $MSE_4$  belonging to  $ME_4$ . Therefore a direct path between A and B processors is established.

The bus considered may organize parallel byte exchanges between computers, when A processor sends to B processor not the entire word but any portion of this word, which is a multiple of 16. This permits communications between different size processors when a smaller size processor B receives a byte from larger size processor A which matches its size. ■

### 5.2.2. Organization of shifts in one computer

Within one computer of the UDC group, one can use the path established above between a pair of processor elements for organizing  $16 \cdot f$ -bit shifts, i.e., those which are multiple of 16. Indeed, existing limitations on the number of pins in LSI modules make it far more difficult to organize various types of parallel shifts of  $16 \cdot k$ -bit words, because any  $m$ -bit shift takes  $m$  pins in each PE. Therefore for  $16 \cdot f$ -bit shifts, i.e., those multiple of 16, it is expedient to use the Memory Processor Bus, since these shifts take no additional interconnections between PE elements. The  $16 \cdot f$ -bit shift of a word is reduced to parallel transfer of each 16-bit byte of this word from  $PE_i$  to  $PE_j$ , where  $j = i + f$  and  $f$  shows how many 16-bit shifts must be performed. Direction of the shift depends on the sign of  $f$ , i.e., for left shift  $j = i - f$ , for right shift  $j = i + f$ .

To organize a 16-bit path from  $PE_i$  to  $PE_j$  one may use the same serial connection of two connecting elements  $MSE_i$  and  $MSE_j$  assigned to the memory element  $ME_j$  which was used for conventional communication between different PE considered earlier. For shifts however, PE's have to perform two actions: send its own byte to a destination PE, and receive a shifted byte from another PE. Since each PE is connected via 16-bit bus with MSE elements, it may perform these two actions only in two clock periods. During the first clock period,  $PE_i$  issues 16 bits to a register of  $MSE_i$  assigned to  $ME_j$ . Activation of this  $MSE_i$  is performed through  $ASE_j$  assigned to  $PE_i$ , when  $PE_i$  sends  $SEL = i + f = j$ , where  $i$  is its position code,  $f$  is the shifting constant stored in the shift instruction. At the second clock period, each PE sends SEL equal to its own position code. Therefore for each  $PE_j$  (which has to receive a shifted byte from  $PE_i$ )  $SEL = j$  selects local  $ASE_j$  and then  $MSE_j$  assigned to  $ME_j$  for broadcasting 16-bit word from  $MSE_i$  to the  $PE_j$ .

In organization of a  $16 \cdot f$ -bit shift, one has to consider that the meaning of  $SEL = i + f$  has to identify positions of those PE's which belong to a computer performing this shift. This means that for a left shift,  $f$  most significant PE's should not form  $SEL = i - f$ , otherwise each of them will transfer a 16-bit byte to a PE contained in the left neighboring computer. Likewise for right shift,  $f$  least significant PE's should not form  $SEL = i + f$ . Otherwise, each of them will transfer a 16-bit byte to a PE contained in the right computer. For each

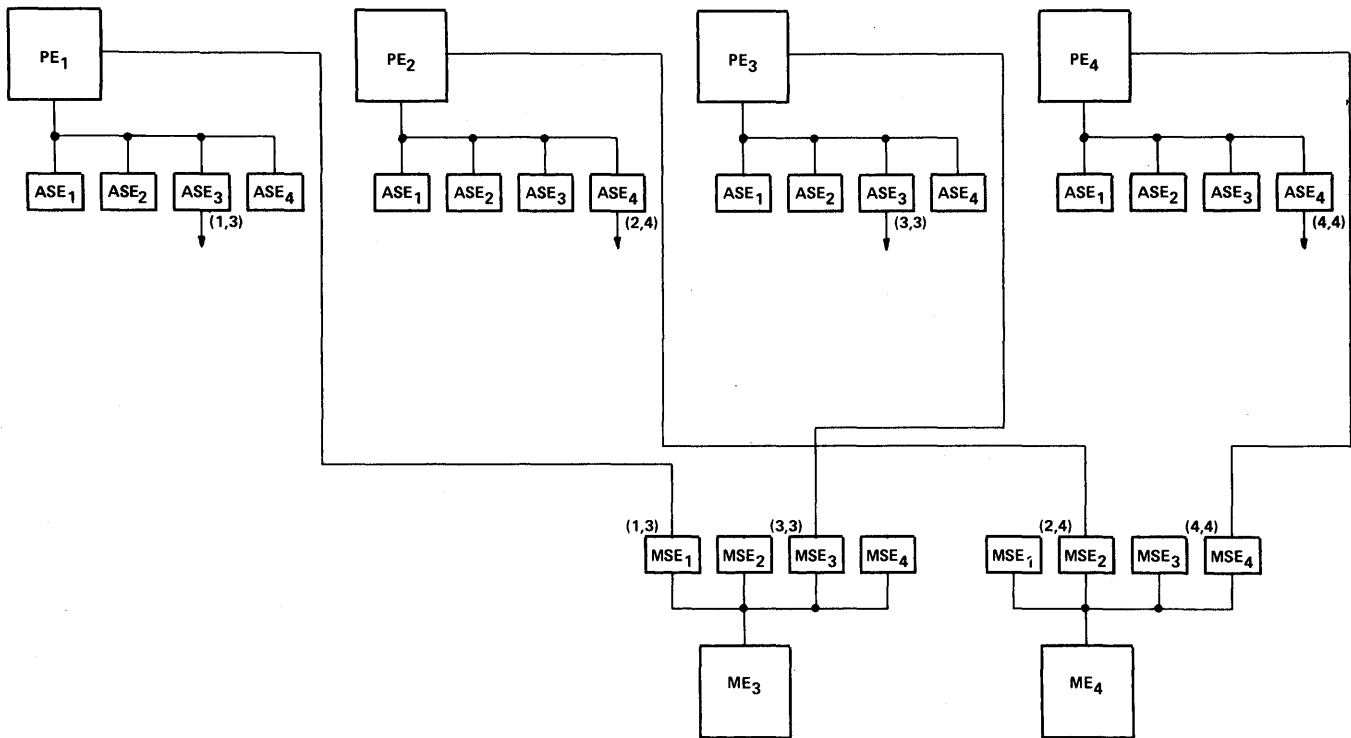


Figure 5—Exchanges between two processors.

computer blocking of unwanted lefthand and righthand PE's from forming  $SEL = i \pm f$  is performed with special constants  $g$  and  $d$ . For  $16 \cdot f$ -bit shifts the test  $i > g$  shows positions  $i$  of PE's which may participate in the left shift, likewise the test  $i < d$  specifies positions of PE's which may participate in the right shift. Using both constants  $g$  and  $d$ , the shift instruction may perform a  $16 \cdot f$ -bit shift of a portion of word restricted by PE<sub>*g*</sub> from the left and PE<sub>*d*</sub> from the right.

*Example 7.* Let the resource be formed into  $C_1(6)$  computer and a 48-bit word stored in PE<sub>2</sub>, PE<sub>3</sub>, PE<sub>4</sub> have to be shifted by 32 bits to the right. Thus  $f=2$ ,  $g=1$ ,  $d=5$ , since left PE<sub>1</sub> and right PE<sub>5</sub>, and PE<sub>6</sub> should be blocked from shifting. When the shift instruction is fetched, all PE's of  $C_1(6)$  computer perform two sequential tests:  $i > g = 1$  and  $i < d = 5$  where  $i$  is position code of each PE. These two tests identify that PE<sub>2</sub>, PE<sub>3</sub>, PE<sub>4</sub> store a word to be shifted. During the first clock period of shifting, PE<sub>2</sub> forms  $SEL = 2 + 2 = 4$ . This selects ASE<sub>4</sub> and 16-bits from PE<sub>2</sub> are written to MSE<sub>2</sub> assigned to ME<sub>4</sub>. For PE<sub>3</sub>,  $SEL = 3 + 2 = 5$  sends 16-bits from PE<sub>3</sub> to MSE<sub>3</sub> assigned to ME<sub>5</sub>. For PE<sub>4</sub>,  $SEL = 4 + 2 = 6$  sends 16-bits from PE<sub>4</sub> to MSE<sub>4</sub> assigned to ME<sub>6</sub>. During the second clock period of shifting each PE of the computer generates SEL equal to its own position code. For PE<sub>4</sub>,  $SEL = 4$  activates ASE<sub>4</sub> and MSE<sub>4</sub> assigned to ME<sub>4</sub>. Therefore 16-bit byte stored in MSE<sub>2</sub> of ME<sub>4</sub> is now broadcasted through MSE<sub>4</sub> to PE<sub>4</sub>. Similar actions are performed by PE<sub>5</sub> and PE<sub>6</sub>. Since for PE<sub>1</sub> through PE<sub>3</sub>, no shifted byte was received by MSE's assigned to their ME's, generation of each  $SEL = i$  in these PE's does not lead to fetching a shifted byte. ■

### 5.2.3. Parallel exchange between the processor and the memory of two computers

For two computers A and B, if A computer needs an array of data words stored in the memory of B computer, the A computer establishes a direct communication path between the A processor and B memory whereby the A processor fetches a second operand from the B memory.

Two types of such exchange are possible.

- (1) Whole  $16 \cdot k$ -bit word exchanges, when sizes of A and B computers match and the A processor fetches a word from the B memory which matches its size. For instance, in Figure 1, there are two 32-bit computers A and B involved in the processor-memory exchanges. Accordingly the A processor (PE<sub>1</sub>, PE<sub>2</sub>) accesses first operands from its own primary memory (ME<sub>1</sub>, ME<sub>2</sub>) and second operands from the memory (ME<sub>3</sub>, ME<sub>4</sub>) of the B computer.
- (2) Byte exchanges, when the A computer may fetch from the primary memory of the B computer not the entire  $16 \cdot k$ -bit word(s) but any portion of this word which is a multiple of 16. Namely, the A computer may access 16, 32, ...,  $(k-1) \cdot 16$ -bit bytes of a  $16 \cdot k$ -bit word.

Since whole word exchanges were considered in [17], this paper will focus on byte exchanges.

There exist two types of byte exchanges, processor mem-

ory, caused by specificities of parallel data broadcasts of variable size words:

First, an A computer size may be smaller than that of B computer, and A computer fetches a byte from B computer which matches the size of A. Such an exchange allows a smaller in size computer to perform a fast access to the words which match its size from the primary memory of larger in size computer.

Second, not all, but several processor elements, PE, of the A computer may access a matching number of memory elements of B computer. This byte exchange allows a larger size processor A to fetch a smaller size word from the B memory. Or the A computer may perform an associative scanning of data arrays with smaller word sizes which are stored in separate memory modules of B computer, etc.

*Example 8.* Let the resource form 48-bit computer,  $C_1(3)$ , and 64-bit computer,  $C_4(4)$  (Figure 6). Suppose that  $C_4(4)$  computer needs that its PE<sub>5</sub> and PE<sub>6</sub> fetch data words from ME<sub>2</sub> and ME<sub>3</sub> contained in  $C_1(3)$ ;  $C_4(4)$  computer uses a special byte exchange instruction which stores the following values: displacement code  $z=2-5=-3$  since ME<sub>2</sub> stores the word, and PE<sub>5</sub> accesses it. Similarly,  $z=3-6=-3$  allows ME<sub>3</sub> to send a byte to PE<sub>6</sub>. Also stored are position codes,  $g$  and  $d$ , of PE's which should receive, respectively, most and least significant bytes of the word. (For our case  $g=5$  and  $d=6$ , since PE<sub>5</sub> and PE<sub>6</sub> receive respectively most and least significant portions of a 32-bit word).

When this instruction is fetched to all PE's of the  $C_4(4)$  computer, each PE compares the  $g$  value received through the instruction with its own position code  $i$ . If  $i < g$ , this PE is blocked from the word fetch. If  $i \geq g$ , the next test is performed. All processor elements which satisfied the first test,  $i \geq g$ , are then tested for  $i \leq d$ . If  $i \leq d$ , the respective PE fetches word. If  $i > d$ , this PE does not fetch the word. Thus two sequential tests  $i \geq g$  and  $i \leq d$  allow one to identify a portion of consecutive PE's of the  $C_4(4)$  computer which should fetch the word from the  $C_1(3)$  computer. According to this rule, in the  $C_4(4)$  computer, PE<sub>4</sub> is blocked from fetch, since PE<sub>4</sub> does not pass through the first test  $i \geq g$  ( $i=4 < g=5$ ). The remaining PE<sub>5</sub>, PE<sub>6</sub>, PE<sub>7</sub> continue the next test:  $i \leq d$ , where  $d=6$ . This test is satisfied only in PE<sub>5</sub> and PE<sub>6</sub>, because  $i=5 \leq 6$  and  $i=6 \leq 6$ . For PE<sub>7</sub>, since  $i=7 > 6$ , it is blocked from data fetch. Thus fetch is performed only by PE<sub>5</sub> and PE<sub>6</sub>. Each of these PE's finds code  $SEL=i+z$ , where  $i$  is the position code,  $z$  is displacement value. For PE<sub>5</sub>,  $SEL=5-3=2$

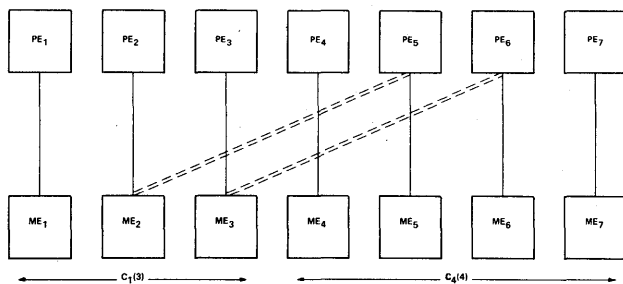


Figure 6—Parallel byte exchanges.

activates ASE<sub>2</sub>. For PE<sub>6</sub>,  $SEL=6-3=3$  activates ASE<sub>3</sub>. This establishes the data path between PE<sub>5</sub> and ME<sub>2</sub> and PE<sub>6</sub> and ME<sub>3</sub> resulting in data fetch from ME<sub>2</sub> and ME<sub>3</sub>, respectively, to PE<sub>5</sub> and PE<sub>6</sub>. ■

Note: the instruction of byte exchange considered above may perform concurrent fetch-shift-operation, i.e., a word fetched from primary memory of one computer may be shifted before being written to the processor of another computer. To do this one has to change the meaning of  $g$  and  $d$  values. Therefore, a merger of multicomputer and multiprocessor architectures accomplished in the UDC group allows both formation of variable size computers and organization of direct and flexible information exchanges between various functional units of these computers.

### 5.3. Array architecture

In the UDC group a portion or the totality of the resource may form one or several concurrent arrays. In each array, one  $16 \cdot k$ -bit processor P\* assumes the function of processor-supervisor and broadcasts instructions fetched from local memory to all other processors working in array. Since P\* contains  $k$  processor elements, PE, its primary memory contains  $k$  ME's. For dynamic architectures to maintain the universality of programs, a 16-bit instruction format is accepted [10,11]. Accordingly, one instruction may be stored in one ME. Thus P\* may store instructions in any of its  $k$  memory elements and broadcast them to all PE's of the array using the same techniques of instruction broadcast which were discussed in [17] for multicomputer architecture, i.e., all PE's of the array store the same program selection code,  $PSE=j$ , which shows position  $j$  of the ME <sub>$j$</sub>  where a currently executed program segment is stored. This insures that an instruction fetched from ME <sub>$j$</sub>  is broadcast to all PE's of the array and prevented from going to other neighboring arrays or computers. A change in PSC leads to a change in memory element from which a new program segment is fetched. This is accomplished by a special Jump ME<sub>1</sub>→ME <sub>$j$</sub>  instruction which organizes fetches of a next program segment from a new memory element. Execution of a similar instruction for multicomputer architecture was considered in [11].

*Example 9.* The array architecture for the UDC group containing  $n=4$  computer elements, CE, is shown in Table III. This architecture may assume array states distinguished by the number of concurrent arrays, and the number and sizes of processors working in each array. For every array, a processor-supervisor function may be assumed by any of its  $16 \cdot k$ -bit processors which then begins instruction fetches from one of memory elements, ME, contained in its local primary memory. In Table III, current positions of computer supervisors are marked with (\*). For state  $N_0$ , one array is formed. If contains four 16-bit processors, P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>, P<sub>4</sub>. This means that each instruction handles a 4-dimensional data vector  $(a_1, a_2, a_3, a_4)$  made of 16-bit words. If P<sub>1</sub> is a processor-supervisor then it broadcasts instructions to P<sub>2</sub>, P<sub>3</sub>, P<sub>4</sub>, i.e.,  $P_1 \rightarrow P_2, P_3, P_4$ .

For state  $N_6$ , one array is formed. It handles a two dimensional data vector  $(a_1, a_2)$ , where  $a_1$  is a 48-bit word

TABLE III.

Code of State	Architectural Configuration	Symbolic Notation of the Architecture
$N_0$		$P_1 \rightarrow P_2, P_3, P_4$
$N_1$		$P_1 \rightarrow (P_2, P_3), P_4$
$N_2$		$P_1 \rightarrow P_2, (P_3, P_4)$
$N_3$		$P_1 \rightarrow (P_2, P_3, P_4)$
$N_4$		$(P_1, P_2) \rightarrow P_3, P_4$
$N_5$		$(P_1, P_2) \rightarrow (P_3, P_4)$
$N_6$		$(P_1, P_2, P_3) \rightarrow P_4$
$N_7$		$P_1 \rightarrow P_2; P_3 \rightarrow P_4$

(\*) means instruction generator.

processed by 48-bit processor ( $P_1, P_2, P_3$ );  $a_2$  is a 16 bit word handled by 16-bit processor ( $P_4$ ). Current position of processor-supervisor is 48-bit processor ( $P_1, P_2, P_3$ )\*. It may fetch instructions either from  $ME_1$ , or  $ME_2$ , or  $ME_3$ . For the  $N_7$  state two concurrent arrays are formed. Each of them handles a 2-dimensional vector ( $a_1, a_2$ ) made of two 16-bit operands. Current positions of processor-supervisors are  $P_1^*$  for the first array and  $P_3^*$  for the second. ■

#### 5.4. Pipeline architecture

The UDC group resources may assume a pipeline architecture called a dynamic pipeline [16,17]. Such pipeline may perform the following adaptations to executing algorithms: (a) adaptation to a sequence of operations, whereby the same pipeline may execute any sequence of operations incurring no time losses for reconfiguring or bypassing some resources as it is done in existing pipelines; (b) dynamic variation of a pipeline's length which means that a pipeline may change

dynamically the number of activated stages so that they match a current number of operations realized in a given program instruction; (c) variable time interval where each pipeline stage may generate minimal time intervals for all operations it executes; (d) on-line feeding of pipeline stages with temporary results they may need for future computations, etc.

Presently the authors are involved in finding techniques for architectural transitions from pipeline architecture to any other architecture(s) (array, multicomputer, multiprocessor) to be assumed by UDC groups. Thus this paper will not consider various pipeline states in which the available resources may be reconfigured.

#### 5.5. Mixed architectures

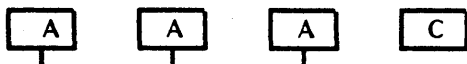

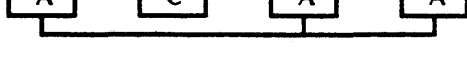
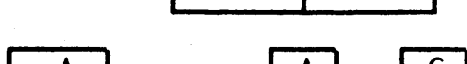


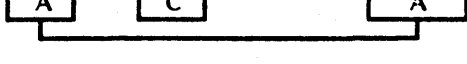
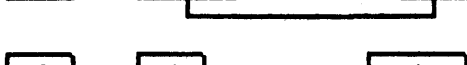





The UDC group resources may switch into several co-resident architectures. Presently, array, multicomputer/multiprocessor may co-exist concurrently in the same system. In

the future, the available resources may reconfigure into co-resident pipelines, arrays, multicomputer/multiprocessors. This allows the same resource to be continuously involved into different types of computations which are presently performed by separate dedicated subsystems. Accordingly, a Supersystem may realize an additional throughput increase using the same complexity of the available resources.

*Example 10.* Table IV shows some of the architectural

states that may be assumed by the mixed architecture for the UDC group containing four CE's. Since multicomputer and multiprocessor architectures merge,  $C_k(k)$  means one  $16 \cdot k$ -bit computer, A means one array. For instance, state  $N_4$  forms one array A and one 16-bit computer  $C_4(1)$ . The A-array includes 32-bit processor,  $(P_1, P_2)$ , and 16-bit processor,  $P_3$ , i.e.,  $A: [(P_1, P_2), P_3]$ . This array handles a 2-dimensional data vector  $(a_1, a_2)$  where  $a_1$  and  $a_2$ , respectively,

TABLE IV.

Code of State	Architectural Configuration	Symbolic Notation of the Architecture
$N_0$		A: $P_1, P_2, P_3; C_4(1)$
$N_1$		A: $P_1, P_2, P_4; C_3(1)$
$N_2$		A: $P_1, P_3, P_4; C_2(1)$
$N_3$		A: $P_2, P_3, P_4; C_1(1)$
$N_4$		A: $(P_1, P_2), P_3; C_4(1)$
$N_5$		A: $P_1, (P_2, P_3); C_4(1)$
$N_6$		A: $(P_1, P_2), P_4; C_3(1)$
$N_7$		A: $P_1, (P_3, P_4); C_2(1)$
$N_8$		A: $(P_2, P_3), P_4; C_1(1)$
$N_9$		A: $P_2, (P_3, P_4); C_1(1)$
$N_{10}$		A: $P_1, P_2; C_3(2)$
$N_{11}$		A: $P_1, P_4; C_2(2)$
$N_{12}$		A: $P_3, P_4; C_1(2)$



are 32-bit and 16-bit data words. State  $N_{10}$  forms a one A-array,  $A:[P_1, P_2]$  and one 32-bit computer  $C_3(2)$ . The A-array contains two 16-bit processors,  $P_1$  and  $P_2$ . ■

In a mixed architecture each array may include processors which are not necessarily adjacent ones. However each  $16 \cdot k$ -bit processor which contains  $k$  PE's and handles  $16 \cdot k$ -bit words is assembled only from adjacent PE's. The reason for this is that in order to perform fast computations a reconfigurable processor bus must be dedicated. However, if it interconnects not only adjacent PE's into a  $16 \cdot k$ -bit processor, the number of possible combinations of PE which may be included into one  $16 \cdot k$ -bit processor grow exponentially. This leads to an exponential increase in the complexity of the bus. On the other hand, should each  $16 \cdot k$ -bit processor be assembled from adjacent PE's, one may construct very simple and fast processor buses which perform fast reconfigurations and take minimal complexity of the hardware [11,17].

## 6. CONCLUSIONS

The appearance of LSI modules with high throughput makes it feasible to organize cost-effective reconfiguration of module interconnections. This allows obtaining of new types of architectures, otherwise called dynamic architectures. A Supersystem with dynamic architecture may realize additional performance gains on the same resource by taking advantage of the following factors: (a) by reconfiguring resource into minimal size computers, it may maximize the number of programs computed by the same resource; (b) by switching the resources into different types of architecture—array, pipeline, multicomputer/multiprocessor—it may speed up respective computations. This allows the available resources to be permanently involved in even those computations that require dedicated subsystems: array and pipelines.

Therefore, the use of dynamic architectures in Supersystems leads to the realization of new sources of throughput increases heretofore unused in traditional parallel systems.

## REFERENCES

1. Bailey, F. R., "Computational Aerodynamics—Illic IV and Beyond," *Digest of Papers*, Spring CompCon '77, pp. 8-11.
2. Vick, C. R., "Research and Development in Computer Technology. How Do We Follow the Last Act," Keynote Speech, *Proceedings of the International Conference on Parallel Processing*, 1978, pp.1-5.
3. Vick, C. R., "A Dynamically Reconfigurable Distributed Computing System," Doctoral Dissertation, The Graduate Faculty of Auburn University, Alabama, 1979.
4. Vick, C. R., Scaff, J. E., and McDonald, W. C., "Distributed Data Processing for Real-Time Applications," *Proceedings of the Sixth Texas Conference on Computing Systems*, 1977.
5. Fitzgibbon, H., Buckles, B., and Scaff, J., "Distributed Data Processing Design Evaluation Through Emulation," *Proceedings Computer Software and Applications Conference (CompSac)*, 1978, pp. 364-369.
6. von Neumann, J., "Probabilistic Logic and the Synthesis of Reliable Organism from Unreliable Components," *Automata Studies*, Eds: C. E. Shannon and J. McCarthy, Princeton University Press, 1956, pp. 48-98.
7. Friedman, A. D. and Saheban, F., "A Survey and Methodology of Reconfigurable Multi-Module Systems," *Proceedings Computer Software and Applications Conference (CompSac)*, 1978, pp. 790-796.
8. Kartashev, S. I. and Kartashev, S. P., "A Multicomputer System with Software Reconfiguration of the Architecture," *Proceeding of the Eighth International Conference on Computer Performance, SIGMETRICS CMG VIII*, Washington, D.C., 1977, pp.271-286.
9. Lipovski, G. J. and Tripathi, A., "A Reconfigurable Varistructured Array Processor," *Proc. International Conference on Parallel Processing*, 1977, pp. 165-174.
10. Kartashev, S. I. and Kartashev, S. P., "Dynamic Architectures: Problems and Solutions," *Computer*, vol. II, July 1978, pp. 26-40.
11. Kartashev, S. I. and Kartashev, S. P., "Multicomputer System with Dynamic Architecture," *IEEE Transactions on Computers*, vol. C-28, no. 10, October 1979, pp. 704-721.
12. Kartashev, S. I., Kartashev, S. P., and Ramamoorthy, C. V., "Adaptation Properties for Dynamic Architectures," 1979 National Computer Conference, *AFIPS Conference Proceedings*, AFIPS Press, 1979, vol. 48, pp. 543-556.
13. Ibbett, R. N. and Capon, P. C., "The Development of the MU5 Computer System," *Communications of the ACM*, vol. 21, no. 1, January 1978, pp. 13-24.
14. Watson, W. J., "The TI ASE—A Highly Modular and Flexible Super Computer Architecture," *In AFIPS 1972 Fall Jt. Computer Conf.*, AFIPS Press, Montvale, N.J., 1972, pp. 221-228.
15. Russell, R. M., "The CRAY-1 Computer System," *Communications ACM*, vol. 21, January 1978, pp. 63-72.
16. Kartashev, S. P. and Kartashev, S. I., "Adaptable Pipeline System with Dynamic Architecture," *Proceedings of the 1979 International Conference on Parallel Processing*, pp. 222-230.
17. Kartashev, S. P. and Kartashev, S. I., "Performance of Reconfigurable Busses for Dynamic Architectures," *Proceedings of the 1st International Conference on Distributed Computing Systems*, Huntsville, Alabama, 1979, pp. 261-273.

# The highly-parallel supercomputers: definitions, applications and predictions

by HUBERT H. LOVE, JR.

Radar Systems Group, Hughes Aircraft Company  
Lincoln, Nebraska

## 1.0 INTRODUCTION

As computer processing power has increased over the past three decades, so have demands on computer performance. In the race of computer technology to keep abreast of these demands, more and more attention has been given to parallel hardware organizational techniques. Instruction and data pipelining, instruction overlapping and distributed processing are examples of such techniques.

More recently, a particular class of parallel architectures, which we shall term "supercomputers," has been receiving special attention. These are the very large, highly parallel reconfigurable array processors. In such a machine, the processing task is distributed among a large number of identical processors which are organized into an array configuration by means of a communication network. With proper algorithms and hardware implementation, these computers can achieve massive throughput rates in a number of useful applications. These include radar signal processing, short-term weather prediction, complex query/response systems and high-speed text processing.

There are several reasons for this increased interest. One, of course, is the importance of the aforementioned applications. Another is the possibility for massive computing capability inherent in the emerging VLSI technologies, and the particular suitability of highly-parallel computer organizations to these technologies. Still another is the increased importance of fault tolerance in computer systems designed for advanced, real-time applications. Many highly-parallel architectures have an inherent fault-tolerant capability.

The body of the present paper describes some of the highly parallel computer organizations and the applications suitable for them. General criteria for applicability are also given. The implications of VLSI with respect to these supercomputers and to fault-tolerant capability are explained. Finally, some predictions and suggestions are made as to the future course of highly parallel supercomputer development and the nature of the applications for which they will be intended.

## 2.0 GENERAL DESIGN CHARACTERISTICS

A reconfigurable array parallel system has several well-defined characteristics by which it can be identified. These are:

### *Multiple processors in simultaneous operation*

Most parallel array systems contain many identical processors (called "processing elements" or "PE's") capable of simultaneous arithmetic and logical operations. Their complexity varies from a few hundred gates to many thousands or tens of thousands. Their capability ranges from basic bit-serial logical and arithmetic operations on a single pair of given operands to full-scale bit-parallel processing on operands selected from large memories. The PE's themselves may contain their own internal memories or may use multiple shared memories, or may have combinations of both. During execution, various PE's will generally process similar data, using a common algorithm. In most systems, the PE's do not contain their own programs, but obtain their common control signals from external control units.

Figure 2-1 is a simplified block diagram of a typical reconfigurable array parallel system. The system shown contains sixteen processing elements.

### *Communication channels for control and data*

All reconfigurable array parallel systems contain communication channels linking the processing elements. In almost all systems the PE's are linked to one another by these channels. In a few, they are simply connected to the common controller processor which determines their operation.

The organization and capability of the system's communication channel network is critical to the system's applicability and performance. These channels pass data status and control information from one PE to another or from one to many PE's. The channels range in complexity from bit-serial to bit-parallel. Each PE may have one or many channels, each of which may connect to one or more of the other PE's or to a central control processor.

In general, each PE exercises some internal control over one or more of the following:

- a) the internal source of the data to be output on the channel,
- b) the particular channel to be involved, if there is more than one,
- c) the intended destination of the data, whether a selected set of PE's or a central control unit,

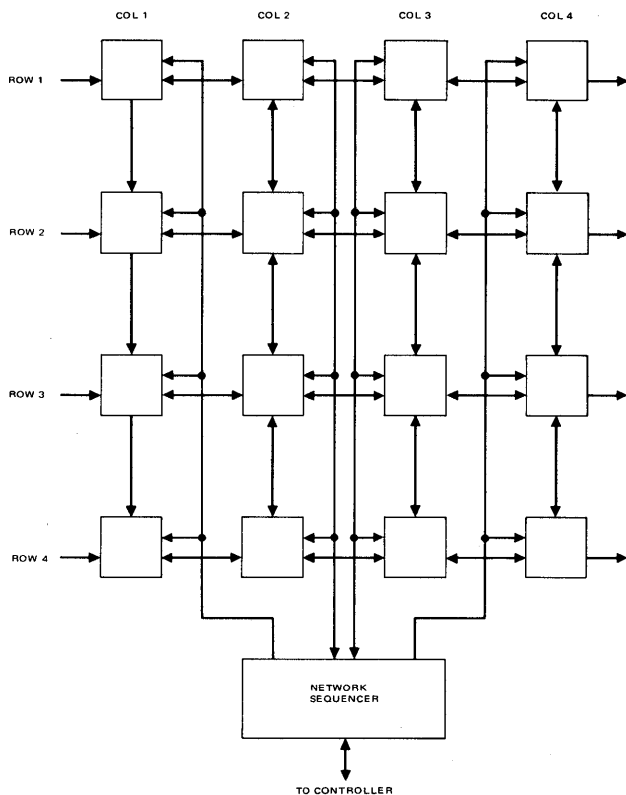


Figure 2-1—Typical reconfigurable array parallel processor.

- d) the internal destination (register, memory location, etc.) of the data received on a channel.

#### *Control of execution: central or distributed*

Control of PE and communication channel operation is generally shared between the individual PE's and some central control unit (CU). The control unit consists of one or several serial processors with special interfaces for monitoring PE status and outputting common data and control information to the PE's. In general, the CU contains the system's programs and the system instruction-decoding logic as well. The individual PE's determine their operating states by a combination of the control signals from the CU and their own internal states, the latter being usually dependent on the results of the processing of the individual PE's particular data.

In Figure 2-1, the control signal paths from the CU to the processing elements are shown as dotted lines. The data paths between the PE's are shown as solid lines, as are the data paths between CU and the PE's. Some communication of data between the CU and the array is required, but will be minimal in a well-designed system.

In a few system designs, there is essentially no central control unit. Instead, control is distributed among the PE's themselves in multiple-instruction streams which execute in parallel. The Holland Machine [3] is an example of this.

#### *Input and output*

Most parallel array systems have multiple input and output channels, the reason being that they are expected to have a high data throughput in most applications. In many systems, there is an input and output channel for each processing element. In the system shown in Figure 2-1, there are input and output channels only for the processing elements on one edge of the array, with the communication network providing for the remainder of the input and output operations.

#### *Fault-tolerant capability*

All parallel array designs share the capability to some degree, even if not actually implemented, to provide graceful degradation of performance in case of hardware failure, and usually the potential capability for software assisted fault isolation and recovery as well. This is inherent in the parallelism of the architecture. It is enhanced by the fact that the majority of the system logic is replaceable in small units, and thus cannot cause single-point failures.

### 3.0 THE APPLICATIONS AND THEIR CHARACTERISTICS

Applications suitable for reconfigurable array processors are not as numerous as for conventional serial processors, but are more common than may at first be supposed.

Present and potential applications include:

1. data base management and query processing, particularly when the data base is complex and dynamically varying, when the queries are very complicated, and when fast response time is a requirement;
2. real-time text processing, again when the processing is complex and fast response is essential;
3. applications dealing with large matrices; weather prediction, determination of neutron flux densities in reactors;
4. real-time radar data processing; track-while-scan, radar pulse deinterleaving;
5. real-time data compression involving very high data rates;
6. image processing, both real-time and non-real time.

For some input and throughput requirements, many of these tasks are quite beyond the capability of serial processors, and also the fast pipeline processors, even with the most advanced gate technologies presently being developed.

#### *Determining the suitability of an application*

There are a number of reasonably well-defined criteria which can determine the suitability of a prospective application for implementation on a reconfigurable array parallel system. These are:

### Parallelism in the algorithms

There are several ways in which parallelism in the application algorithms can be present.

First of all, some applications have a characteristic which shall be termed "block oriented." Block-oriented applications are those which deal with a number of similar "objects" in the data base or in the outside world, generally a varying number, and in which the identical general process is performed on all of them. In such applications, a block of memory or a processing element in the processor will be assigned to each such "object," and will contain the parameters and working space for that object. The "objects" may be:

1. targets or threats being tracked by radar;
2. blocks of raw English text being searched or modified;
3. records in a file being searched or modified;
4. pixels or sets of pixels in an image being processed.

Depending on the particular architecture, separate PE's will be assigned to each block, a single PE will process several blocks, or a number of PE's will be assigned to each block. The determination of the assignment is made dynamically during processing in some systems.

### Complexity of the process

The existence of parallelism in the application is not sufficient justification for the use of parallel hardware. It is a requirement that the amount of processing on each piece or each set of data be sufficient to justify the time required to load the data. If this is not the case, the complicated parallel hardware is not being effectively used, and the processing could perhaps be better performed by other means.

An example for which the use of parallel systems can be well justified is the inversion of a matrix. The matrix need be loaded only once, and the number of operations performed to invert the matrix is very large indeed. Another example is dictionary lookup using an associative memory (which will be defined later), when the same dictionary resides in the memory during a large number of lookups and when the dictionary entries are of variable size.

### Input considerations

Parallel systems are most efficiently used when the amount of processing is high in proportion to the quantity of input data. The actual input data rate which can be accommodated varies considerably with the system design. Some systems, having very powerful bit-parallel processors as PE's, can handle correspondingly high input rates. Other systems, such as bit-serial associative processors, which may have processing capability as great as the former, may not be able to handle such high input rates. This capability is of major importance in choosing a parallel system architecture.

### Output considerations

There is a point of considerable importance regarding output rates for parallel systems in real-time applications which deserves special mention. Parallel systems have very high internal processing capability and can accommodate very high input data rates with the reservations just mentioned. However, they are sometimes criticized for having relatively low output data rates and thus being handicapped in real-time applications. This is especially true with regard to the associative processors, which often are able to output no more than a single word slice or bit slice with each clock cycle. (No matter if the word be very long, the rate is still slow in comparison with the internal processing speed.)

In a real sense this criticism is largely unjustified. A purpose of a powerful real-time data processing system should be to reduce the amount of data from a high input rate to a relatively low output rate as a result of its internal processing. But suppose that it does not reduce the rate and that the output rate is as high in proportion to the processing power as for ordinary serial systems. It is difficult to see what could be done with this gargantuan stream of output. Humans cannot absorb it in real time, even if it could be printed fast enough. If the system output is to be directed into another computer for further processing, something is lacking in the system engineering. That is, the justification for the use of a parallel system is presumably its high processing rate, yet it is not doing the entire processing task. It should have been designed to accomplish the entire processing task in the first place.

If the output is to a high-speed telemetry channel, rather than directly to another processor, there is still little justification for massive output capability in general. Reconfigurable array parallel systems can do an effective job of compressing data internally, using one of a number of techniques. Such capability can generally be designed into the system, and the very high output rate (and the need for expensive wide-band telemetry) thus avoided.

### Other parallel attributes

For some system designs, it is possible to organize the processing so that some time-consuming operations can be performed for several different (and even unrelated) processes in parallel by a single hardware operation. This is true for systems, such as most associative processors, which are bit-serial in operation and which depend for their high processing throughput on a high degree of parallelism, rather than on being able to perform individual operations rapidly. Multiplication, division and floating-point arithmetic are examples of operations which are slow on bit-serial associative processors, but whose number of executions can be reduced by making them multi-purpose.

### Communication channel considerations

For many applications, the capability to communicate large amounts of data is critical to system performance, and

the communication scheme must be given special attention in the design. The result of deficient communication capability is an unbalanced system, in which the individual PE's can operate independently with high throughput, but are greatly slowed down when they must communicate with one another. Communication problems are the most difficult to handle when one is trying to determine the suitability of an application for implementation on parallel systems.

#### Avoiding serial operations

In determining the suitability of a task for implementation on a parallel array system, it is not sufficient to demonstrate only that critical operations can be performed in parallel. It must also be shown that all sequences of operations can be performed in parallel fashion without the necessity of serial processing in between. If this is not possible, most of the advantage of the parallel-processing capability will be lost. It takes surprisingly few serial operations to seriously compromise the entire processing scheme. Associative processors are particularly subject to problems of this sort. In a typical case, after a parallel operation is performed, it may become necessary to reorganize the data so that the next operation can be performed in parallel. If the reorganization cannot itself be performed in parallel, the hardware and/or data organization must be redesigned. If this will not suffice, the parallel-processing approach may have to be rejected.

#### 4.0 THE IMPLICATIONS OF VLSI FOR HIGHLY PARALLEL SYSTEMS

Array architectures are complicated and generally have a higher gate count in proportion to their processing throughput than serial architectures. This has hindered their acceptance in the past. However, with the recent Very-Large-Scale-Integration (VLSI) technology, it appears that not only does the handicap of complexity for array systems tend to diminish, but that such systems are even preferable in many respects from the standpoint of fabrication as compared with conventional systems.

#### *Amenability to on-chip fault repair*

Well-designed array architectures consist largely of repeated logical elements. Depending on their complexity, a number of these elements can be fabricated on a single LSI or VLSI wafer. This can mean a higher yield if discretionary wiring or other techniques can be used to disconnect faulty elements during the manufacturing process. Moreover, with proper circuitry, it is often possible to implement software-controlled fault-isolation and fault repair on the wafer itself. This is of considerable benefit in some applications and will be of increasing benefit in all applications as growing system complexity increases the probability of run-time failure. The advantage of repeated circuit elements from the aspect of fault tolerance results from the fact that single failures on

a wafer, or even multiple failures do not render the wafer worthless if the wafer contains spare elements which can be switched into the system under software control.

#### *Suitability for very fast logic*

The two principal characteristics of the gate technology that is presently being developed are extremely high speed (the order of 500 psec. gate delays) and very large numbers of gates per wafer (several hundred thousand). This sounds like very good news at first glance. However, when one looks at the problems being faced by the engineers trying to fabricate this circuitry and design systems using it, a quite different picture emerges.

Many of the biggest problems with such high-speed logic are caused by the "off-chip" delays for signals passing from wafer to wafer. Particularly damaging are the cases involving "round-trip" signals. A round-trip signal is one which passes one way between two chips, causing a signal to be generated at the receiving chip which then passes the other way between the chips. Time and phase lags occur because of such signals, requiring that the clock rate for the system be greatly reduced.

Many parallel-array system designs appear to offer advantages with respect to these problems as compared with more conventional systems. For one thing, inputs to the individual PE's are often very orderly, consisting mostly of control signals originating in logically similar registers within the controller. Since almost all of the processing for the system takes place within the array of processors, there is very little communication of data from the PE's to the controller, thus reducing the occurrence of round-trip delays.

#### 5.0 SURVEY OF PROCESSOR DESIGNS

In this section, two types of general reconfigurable array parallel systems will be described and analyzed. For each type, a general description of the system, its operation and applications will be given and comparisons with other system approaches made.

The system types to be covered are:

1. parallel network processors;
2. associative processors.

#### *Parallel network processors*

The term "parallel network processor" shall refer to a reconfigurable array parallel processor in which

1. the processing elements themselves are full-fledged serial processors with sizable local memories;
2. the communication network connects each PE to several other PE's in a regular fashion;
3. one or several sets of PE's each execute essentially the same program on different sets of data simultaneously

under central control. Generally, there is only one set and one program.

The concept of the network processor arises from the nature of the applications which are involved. These applications deal with "objects" in the real world which interact in the application in some orderly fashion. The communication network in the network processor interconnects the PE's in the same fashion, under software control, and each PE corresponds to one object.

Applications well suited to parallel network processors include:

1. the solution of sets of partial difference equations (by the relaxation technique, for example);
2. general matrix operations;
3. many image-processing applications; each PE corresponds to a pixel or a set of pixels;
4. radar data processing; each PE corresponds to a target or threat being tracked.

#### General organization and operation

The processor organization to be described is essentially the same as that for the SOLOMON I computer, which is the earliest of the well-known parallel network architectures. The ILLIAC IV, probably the most powerful supercomputer yet built, is another example.

#### The processor array

Figure 5-1 shows the organization of the array of processing elements (PE's) for the system. Only 16 PE's are

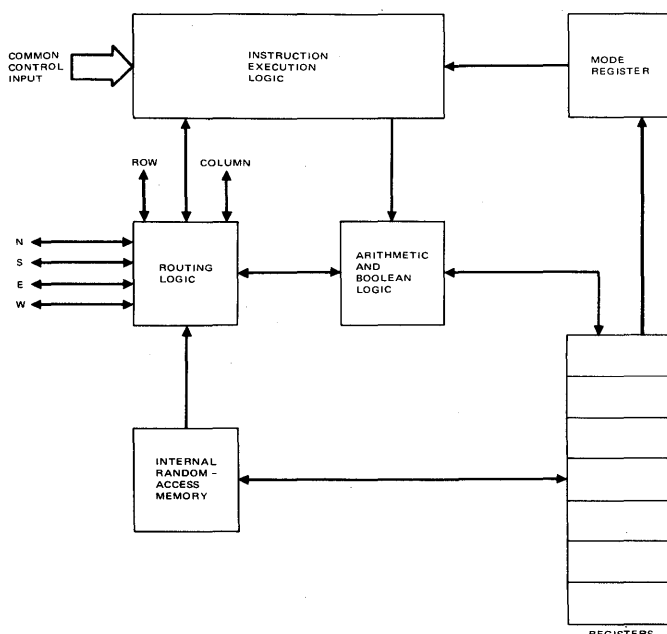


Figure 5-1—Parallel network processor array organization.

shown. The SOLOMON I design has 1024 PE's, organized into 32 rows and 32 columns. The array contains about 90 percent of the entire system. Each PE is a serial processor in every respect except for the absence of instruction-sequencing and decoding logic. The array hardware interconnects the PE's in a two-dimensional array configuration. In the original SOLOMON design, the PE's were bit-serial processors, and the communication channels were also bit-serial. In newer designs, because of the advances in LSI fabrication techniques, the PE's can be bit-parallel. Therefore, in order to maintain system balance, the communication channels are made bit-parallel also.

The operation of the communication channels during program execution is under software control. The control is largely central with respect to the general interconnection modes. These modes permit inter-element communication by one or more of the following schemes simultaneously:

1. Each PE is connected to its two neighbors in the same row. The first PE in each row is connected end-around to the last PE in the row, resulting in a "horizontal" cylindrical configuration.
2. Each PE is connected to its two neighbors in the same column, with the connection end-around as before, resulting in a "vertical" cylindrical configuration. Both this option and the first can be used together.
3. The PE's can all be connected in a single one-dimensional array, end-around if desired.

In addition to the array interconnections, all of the PE's are connected to common buses through which they can receive common data items simultaneously. There are common buses for each row and each column of the array. With this scheme, many common data items can be input to the array simultaneously, a different one for the PE's in each row or each column.

#### The processing element

Figure 5-2 shows the organization of each processing element in the array. Each PE contains a set of working registers and a local random-access memory for data only. There is a logic module for performing arithmetic, Boolean and comparison operations. There is also instruction-execution logic, but there the similarity to a conventional serial processor ends. The PE receives its instruction control from an external source. It has no internal instruction sequencing or instruction decoding logic. In addition, the PE has several unique devices with functions as follows:

1. Routing logic. This controls the source or destination of data and control information passed between the PE and other PE's and common buses. During the execution of an instruction, the routing logic state permits the PE to send or receive information from the common row or column bus.
2. Mode register. This register (which is two bits long in the SOLOMON computer) is set from within the PE.

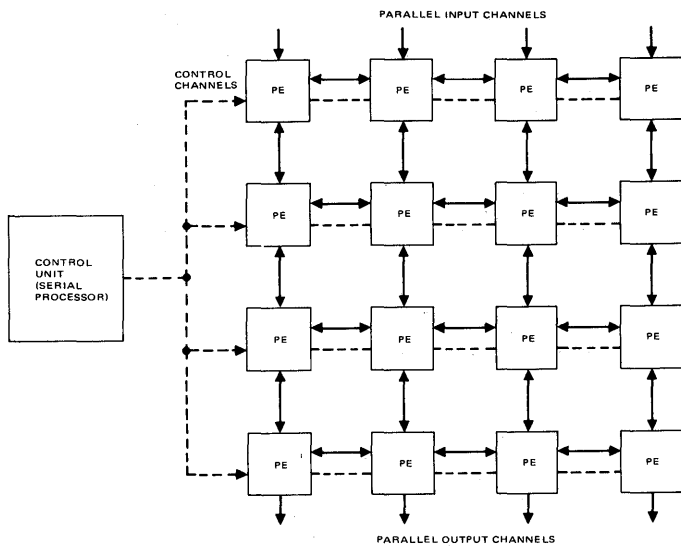


Figure 5-2—Processing element for parallel network processor.

The instruction control information received by the PE for all instructions contains a specification as to which mode or modes the PE must be set in order that the PE execute the instruction. (All modes may be specified, in which the case the execution is mandatory.) Through this device, each PE can determine in most cases whether or not it will execute an instruction. Each PE makes this determination as the result of the processing of its local data, and then sets its mode register accordingly.

The system organization

Figure 5-3 shows the general organization of a parallel network processor. This system consists of the parallel network processor array just described, plus a number of modules for control of the array, for operator interface and for input and output. These modules and their functions are as follows.

1. The Control Unit. This unit receives from the control memory each instruction executed by the system. It decodes the instruction and the addresses. For those instructions to be executed by the PE array, it sends the resulting control information and the row and column addresses to the PE array. These instructions will be executed by those PE's in the specified row(s) and column(s) whose internal mode register settings correspond to the mode settings specified in the instruction.
2. The Control Memory. This memory contains the system's programs, plus common data items and common working storage. The Control Unit has normal serial processing capability, which is used occasionally to calculate common data values and to determine the flow of the instruction execution sequence.

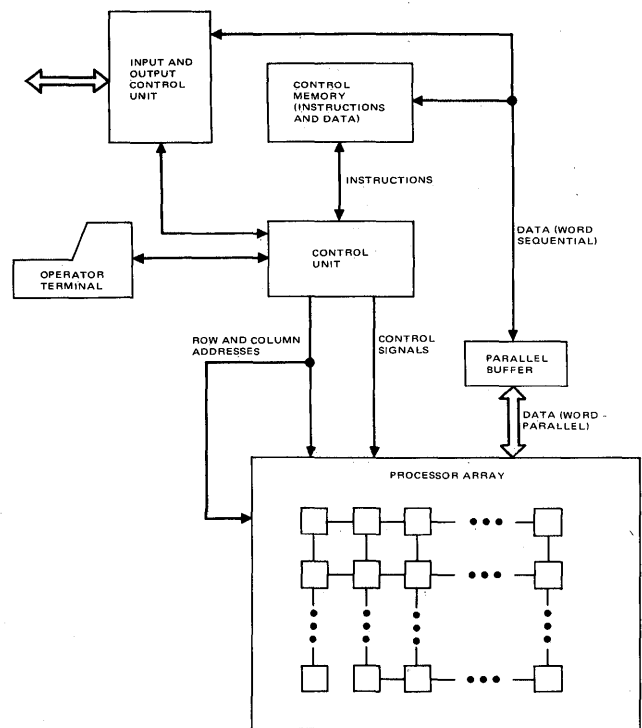


Figure 5-3—Parallel array processor organization.

3. The Parallel Buffer. This buffer receives and sends data in parallel to and from all PE's in the array, or to and from all PE's in specified rows and columns. The buffer holds one data item for each row and one for each column. Data transfer between the Parallel Buffer and the remainder of the system is word-serial.
4. The Input/Output Control Unit. This device controls all system input and output for most applications. For some real-time applications, it is best to bypass such complex control units altogether and input data directly into the processor array.

ILLIAC IV and PEPE

The ILLIAC IV is perhaps the best known example of an array processor. It was designed with very demanding real-time applications in mind, such as real-time radar processing with a phased-array antenna system. The ILLIAC IV processing array consists of 256 large-scale processors having memory cycle times and add times of 240 nsec. (64-bit operands) and multiply times of 400 nsec. The array is essentially two-dimensional, like that of the SOLOMON.

The PEPE (Parallel Element Processing Ensemble) is a highly parallel organization lacking an inter-processor communication network. Instead, each processing element has the capability for independent decision, based upon its internal state, as to whether to participate in a particular operation or sequence of operations. It is also well suited to radar data processing applications, since such applications

require little interchange of information between processors if the program is suitably organized.

### Associative processors

The associative processor is one of the more unusual of the reconfigurable array systems, because its basic operating principle is in a sense the reverse of that for serial processors. The associative processor is based on a device known variously as the "associative memory" or "content-addressable memory" which, in turn, is an outgrowth of a simpler device known as a "search memory."

#### The search memory and its operation

The search memory can be said to function in reverse fashion with respect to a random-access memory. That is, a RAM accepts the address of the desired memory location as input and outputs the contents of that location in response. The search memory accepts the desired contents of the memory location or locations as inputs and outputs flag settings indicating the memory location or locations having those contents. To prevent the operation from being trivial, the content search is field specified.

The distinction between the search memory and a conventional memory is that the search is performed in the former at all locations or cells in the memory simultaneously. This capability is the result of having compare logic at every cell in the memory. (The term "cell" and "word" are sometimes used interchangeably in the discussion to follow, since the cell, which is a hardware device, contains a "word" of data in the usual sense.)

Figure 5-4 illustrates the search-by-content operation as performed by a search memory. In the figure, the memory is shown containing an indefinite number of cells, each containing seven characters (used instead of binary bits for clarity in the illustration). With each cell is shown a corresponding "match flip-flop" which is a flag indicating the success/fail status of the cell as a result of a content search. At the

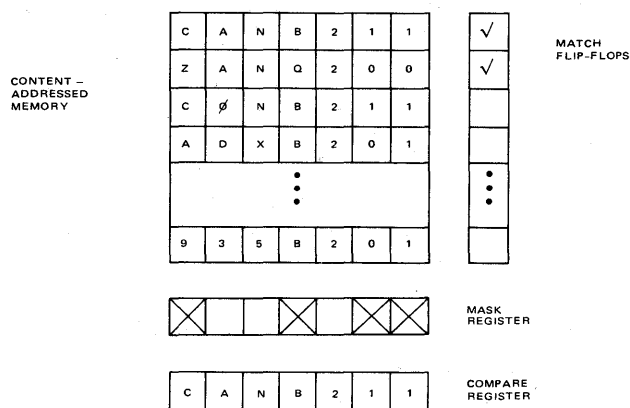


Figure 5-4—Basic associative memory operation: the parallel search-by-content.

bottom of the figure are shown two registers. One of these, called the "compare register" or "comparand register," contains the data item that is the object of the search. The other register, called the "mask register," specifies by its contents the field or fields within the memory cells at which the content search will be made. In the figure, the 1st, 4th, 6th, and 7th character positions are masked out. The search will be conducted only at the remaining character positions.

It is seen that only the first and second cells contain the same characters in the unmasked character positions as the compare register. As a result of the content search, then, the match flip-flops will be set only at those two cells as shown. For most implementations of search memories, the cells' contents reside in a shift register, and bit-serial logic is used at each cell, rather than bit-parallel logic. The operation is still word-parallel, and the desired speed advantage will be achieved if there is enough data to be searched to require a large number of cells.

#### The associative memory and its operation

The associative memory is an extension of the search memory. Its operation is based on the same search-by-content function implemented in the search memory. The associative memory contains additional logic at each cell which permits it to perform word-parallel logical, arithmetic, input and output operations, generally in bit-serial fashion. The logical functions to be described are not common to all associative memories. However, they are quite typical, and give a good characterization of the associative memory and its general capabilities.

#### The parallel-write operation

This operation is that of simultaneously modifying selected bit positions at all cells in the associative memory or at a selected subset of the cells. The bit positions to be modified are determined by the contents of the mask register. The contents to be inserted into those bit positions are the settings of the corresponding bit positions in the compare register and are the same for all selected cells. The cells to be modified are selected by the settings of the match flip-flops.

It is the parallel-write operations and their extensions and variants which give the associative memory its capability to process data internally. This capability includes word-parallel arithmetic and logical operations as well as file searching and modification.

Parallel arithmetic operations are generally implemented through special arithmetic logic, including a full adder, at every cell. Only a single adder per cell is required for a bit-serial machine employing a shift register to contain the cell's data.

#### Other operations

It is in the nature of associative processors to have no conventional hardware addressing for randomly accessing



individual cells. Such serial processing as is necessary is usually limited to that of assessing a selected subset of cells in order (say, a set of cells whose contents have matched to a sequence of searches) so that their contents can be output a word at a time. All associative memories have serial-access logic for this purpose.

Other basic operations implemented in the typical associative memory are such auxiliary and support operations as setting or resetting all match flip-flops and ladder flip-flops, and the operation of outputting the contents of the (single) cell having its ladder flip-flop set.

**The associative processor**

Figure 5-5 shows the organization of a typical associative processor. An associative processor is a system which consists of an associative memory, together with a serial processor for controlling the operation of the associative memory, for controlling input and output, and for interfacing with an operator.

**Reconfigurable associative processors**

It will be noticed that the associative processor as just defined has no reconfigurable characteristics and no inter-communication network (except for the sequential-access logic). For this reason, it is very limited in its applicability. For example, it is impossible in the system just described to transfer data from one cell to another except by means of the word-serial input and output operations using the sequential-access logic or parallel-write logic. For some file-managing operations this is sufficient. However, in order that the associative processor have wider applicability, a communications network must be added so that data and also control states can be transferred between cells simultaneously (that is, between many pairs of cells or from one

cell to many others). A number of techniques for accomplishing this through specialized hardware operating under software control have been devised. Two of these schemes are discussed in the remainder of this subsection.

**The STARAN**

More effort has been devoted to the design, fabrication and application of the Goodyear STARAN\* than to any other associative processor. A number of full scale STARAN systems of various designs have been built, both for general-purpose application studies and for particular designated applications. Some 155 different application functions have been studied for implementation of STARAN, and some 75 of these actually programmed and demonstrated. These deal with such general applications as surveillance systems, sensor signal processing, general scientific applications, communications processing and data management.

This extensive effort has resulted in considerable refinement of the original STARAN concept, and particularly in the addition of a communication network, called the "flip network." The flip network provides the means for performing any desired permutation on the contents of the cells in a 256-cell STARAN module. The STARAN is especially adept in operations in which this fast permutation capability can be effectively utilized, for example, in performing the Fast Fourier Transform (FFT). Other applications include the processing of image data, in which the STARAN, operating as a peripheral device, performs high-speed correlations for a serial processor.

**The ALAP**

The Associative Linear Array Processor (ALAP) [6] uses a different means for obtaining intercell communications from that used by STARAN. The ALAP has a multi-use communication channel, called the chaining channel. Each cell in the ALAP memory array is connected to the next and previous cells in the array by this channel, over which both data and status flags can be transferred. The chaining channel thus organizes the cells into a linear array. Parallel pairwise arithmetic operations can be implemented using the chaining channel.

Applications for which the ALAP has been programmed include radar signal sorting, several radar track-while-scan tasks, and generalized data-retrieval from a structured data base. The ALAP is particularly suited to tasks which are block-oriented in the sense previously defined, and in tasks in which many diverse parallel operations take place simultaneously within the same memory array.

**The massively parallel processor (MPP)**

A very powerful associative processor with an array of 128 x 128 cells is a recent design of Goodyear Aerospace

\* TM, Goodyear Aerospace Corporation, Akron, Ohio.

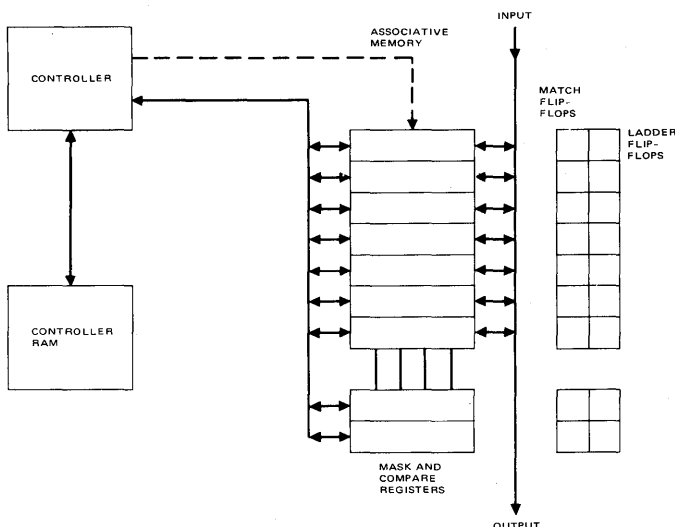


Figure 5-5—Basic associative processor organization.

Corporation. The cells in the MPP are organized into a two-dimensional array by the communication network, with each cell communicating directly with its four nearest neighbors. Each cell can perform arithmetic and logical operations simultaneously with all others, with one operand resident in the cell, and the other either resident in the cell or input from another cell or a common bus. Each cell, in addition to its arithmetic register and arithmetic unit, contains a 1024-bit local memory. All operations within the cell are performed bit-serially.

In the construction of the MPP, almost all components of a sub-array of  $2 \times 2$  cells are fabricated on a single VLSI wafer, using CMOS/SOS technology. The local memories for the cells are constructed from standard  $4 \times 1024$ -bit RAM chips. The clock rate of the array is 10 MHz. Software-controlled fault isolation is provided for each subarray of  $128 \times 128$  cells.

The square array organization is well suited to the intended application of the MPP, which is the processing of satellite image data. From 100 to 10,000 operations per pixel are required for the processing. This satisfies one of the most important of the basic requirements for applicability for parallel processing previously discussed. The two-dimensional communication network will give the MPP an order-of-magnitude increase in processing speed for matrix operations as compared with the STARAN or ALAP. The latter two machines are limited to an order of  $n$ -to-1 increase in speed for such operations as compared to serial processors, where  $n$  is the dimension of the (square) matrix. The MPP will have up to an  $n^2$ -to-1 increase.

## 6.0 FUTURE TRENDS IN RECONFIGURABLE ARRAY DESIGN

To assess the future of very large, fast reconfigurable arrays as practical data-processing systems, one must examine:

1. the expected technology, its advantages, the desirable characteristics of the circuitry that is to be implemented with it, especially with regard to VLSI;
2. the expected applications that will require very powerful array systems;
3. the limitations on the performance of present array processors with respect to throughput, fault tolerance, amenability to VLSI implementation and suitability to the expected applications.

### *The VLSI technology*

The problems caused by the use of several interconnected wafers in a system when the very fast, new logic technologies are used have been mentioned in Section 4. There is another set of problems which have to do with the nature of VLSI itself.

With regard to VLSI, the problem of most concern to system architects is that of taking advantage of the capability

to put so many gates on a single wafer. Conventional serial processors are not altogether suitable for VLSI implementation. One of the main reasons is that it is difficult to factor the logic so that the interfaces between wafers in the system are "clean." This means relatively few pins (not easy when the wafer contains so many gates), similarity in the signal paths to other wafers, and so on. When VLSI fabrication is combined with the very fast gates, the problems multiply. Even interfacing with a conventional RAM when it is on a different wafer from that containing the CPU can be a major problem with 250 MHz. clock speeds.

Some parallel array designs have a strong appeal to the VLSI designers because the number of pins on the wafer is low, and is independent of the number of cells on the wafer, and because almost all of the pins are for bit-serial inputs from identical external registers.

On the negative side with regard to the use of array architectures as candidates for VLSI implementation is the fact that these processors are not yet as general-purpose in their applicability as serial processors. It is evident from the standpoint of economics that wide applicability is a prime requirement for a VLSI wafer which will be very expensive to design. Array processors must therefore be designed with such versatility in mind, and an equal effort must be expended on applications analysis to discover new, highly parallel procedures for many common applications. One promising aspect of some parallel array organizations is that the controllers for the arrays can operate at much lower clock rates than the very fast rates for the arrays, since they have much less to do. This means that serial controllers, with their relatively irregular logic organizations, will not have the off-chip delay problems or the other problems associated with the extremely fast logic needed for the arrays.

### *Future applications for array processors*

The problem of versatility with regard to the applicability of highly parallel array processors has been mentioned. Consider the application areas for which the arrays are suitable or superior. These include parallel file-processing tasks, such as radar data processing. They also include real-time text processing, although suitable systems will be very special-purpose designs. Advanced image-processing tasks, which are important as well as challenging, are high on the list. Very advanced file-retrieval applications, such as question-answering tasks involving deductive and inductive inference, are even more promising tasks, now that suitable algorithms and data structures are being developed.

The author's favorite application field is the aforementioned advanced question-answering systems. Such systems may open the door to such capabilities as the translation and interpretation of context-sensitive languages, and thus to the simulation of human reasoning processes. To accomplish even a very small step in such a direction is an objective of immense appeal. The context-sensitivity of the query-answering process seems to be a key factor. The interpretation of context must require, among many other things, a very large data base of possible contexts which can be searched

very frequently and rapidly with complex search criteria. The potential applicability of very powerful associative processors to such a process is evident.

#### *Limitations in the performance of present designs*

##### **Throughput limitations**

The  $n$ -to-1 limitation of one-dimensional processor array organizations on matrix operations has already been discussed. The MPP, which has a two-dimensional array, overcomes this limitation for many classes of problems. When the structure of the application (for example, the structure of the real-world system being modeled by the computer) does not match the regular array structure, however, such machines suffer a great loss in efficiency of operation.

##### **Fault-tolerance limitations**

In most parallel processor arrays having a regular communication network, there is a tradeoff between the order of parallel operation and the level at which fault tolerance can be implemented. With one-dimensional arrays, fault tolerance can be implemented at the level of the single cell. With the MPP, an entire column of processing elements must be disabled if a single cell fails. This is not appealing in concept as a remedy, and is a most difficult problem with regard to effective fault tolerance.

##### **Limitations with regard to VLSI**

The dimensionality of the processor array is of great importance with regard to amenability to VLSI fabrication. The PEPE, STARAN and ALAP are well suited in this regard. The number of pins on a wafer is independent, or nearly so, for such zero and one-dimensional arrays. For two-dimensional arrays in general, this is not the case. If many cells are fabricated on a wafer, the number of pins will increase as the square root of the number of cells if the array is two-dimensional. All of the cells on the edge of the wafer must have external connections, else processing throughput will suffer because of bottlenecks in the interwafer communication.

A possible solution to this problem is to construct three-dimensional arrays by putting a set of "two-dimensional"

wafers together in a stack like pancakes, with vertical communication channels fabricated directly on pads on the wafers. This may actually be practical at some time in the future, when new technology can be devised, but does not appear to be a near-term solution.

#### *Recommendations, predictions and suggestions*

For a solution to many of the problems just addressed, one is tempted to look at the Holland machine concept. This machine has a two-dimensional array organization. However, in this concept, the communication paths taken by the data in operation are data-determined, rather than location-determined. The Holland machine has serious drawbacks as a practical system. Nevertheless, it does not suffer from some of the limitations with regard to fault tolerance and pin count if it is properly implemented. Perhaps the user should not expect optimal efficiency in the utilization of the hardware. There can be cost/efficiency tradeoffs which can achieve equal results in throughput, while still retaining some of the advantages of the multi-dimensional array. It is these advantages which should be a prime objective of future efforts in parallel array design.

#### REFERENCES

1. Hobbs, L. C., et al., ed., *Parallel Processor Systems, Technologies and Applications*. 1970, New York, Spartan Books.
2. Cannell, M. H., et al., *Concepts and Applications of Computerized Associative Processing, Including an Associative Processing Bibliography*. December, 1970, U. S. Department of Defense Communications, Document No. AD879281.
3. Holland, J. H., "A Universal Computer Capable of Executing an Arbitrary Number of Sub-Programs Simultaneously," *1959 Proceedings of the Eastern Joint Computer Conference*, pp. 108-113.
4. Slotnick, Daniel L., et al., "The Solomon Computer," *Proceedings, 1962 Fall Joint Computer Conference*, pp. 97-107.
5. Barnes, George H., et al., "The ILLIAC IV Computer," *IEEE Transactions on Computers*, Vol. C-17, No. 8, August 1968, pp. 746-57.
6. Finnila, Charles A. and Love, Hubert H., Jr., "The Associative Linear Array Processor," *IEEE Transactions on Computers*, Vol. C-26, No. 2, February, 1977, pp. 112-125.
7. Batcher, K. E., "The Massively Parallel Processor (MPP) System," *Proceedings, AIAA Computers in Aerospace Conference II*, 1979. To be published.
8. Batcher, Kenneth E., "The Flip Network In STARAN," *Proceedings of the 1976 International Conference on Parallel Processing*, 1976, Long Beach, Calif., IEEE.
9. Love, H., Jr., "A Modified ALAP Cell for Parallel Test Searching," *Proceedings of the 1977 International Conference on Parallel Processing*, 1977, Long Beach, Calif. IEEE.

# Database machines and some issues on DBMS standards\*

by STANLEY Y. W. SU,  
*University of Florida*

HSU CHANG,  
*IBM*  
Yorktown Heights

GEORGE COPELAND,  
*Tektronix*

PAUL FISHER,  
*Kansas State University*

EUGENE LOWENTHAL,  
*MRI Systems*

and

STEWART SCHUSTER,  
*TANDEM*

## INTRODUCTION

There are several co-related activities in the database area and computer architecture that make the discussion of database machines and their implications on DBMS standards timely and meaningful. First, in the database area there is a drive toward more powerful database management systems which support high-level data models and languages. The motive for this drive is the requirement to greatly improve user/programmer productivity and to protect applications from changes in the user environment. However, supporting these interfaces with software means often introduces inefficiency in database management systems because of the many levels of complex software which are required to map the high-level data representation and languages to the low level storage representation and machine codes. Second, the need for systems which handle very large databases is increasing rapidly. Very large databases complicate the problems of retrieval, update, data recovery, transaction processing, integrity, and security. Software solutions to these problems work well for both small databases supporting many applications and large databases supporting only a few applications. However, the labor-intensive cost, time delays and reliability problems associated with software development and maintenance will soon become prohibitive as large and highly shared databases emerge. The search for hardware solutions to these problems is a necessary and viable alternative for balancing functionality and price/performance. Third, the progress made in hardware technology in the past decade is phenomenal. The cost of memories, pro-

cessors, terminals and communication devices has dropped and will continue to drop at a drastic rate. It is time for a reevaluation of the traditional role of hardware and software in solving problems of today and tomorrow in database management.

Fourth, there is a vigorous drive toward DBMS standards led by NBS (26,27) aiming to "1) protect the federal investment in existing data, programs, and personnel skills, 2) improve the productivity and effectiveness of database systems available to federal agencies, 3) assist federal agencies with guidelines on the selection, procurement, use, and availability of database systems, 4) perform the research necessary to identify future federal needs and to foster the development of necessary database tools."

Research on database machines is relevant to the study of DBMS standards in the following ways. First, when a standard is to be proposed for adoption it is important to consider how easy the standard can be implemented and the cost involved in its implementation. Database machines may drastically change the ways database management functions are implemented and new technologies may alter the picture of cost involved in database management. A standard is not practical unless it can be implemented with efficiency and reliability. Database machines hold promise to provide more efficient and reliable ways to implement the database functions. Second, very often several alternative designs (e.g. data models or data languages) exist and can be the candidates for standards. Good evaluation and proper selection of these alternatives based on criteria such as "user/programmer productivity," "ease of use," "natural to the user and DBA," etc., are extremely difficult to obtain. In this situation, the selection of one of the alternatives as the standard can be based on, among other variables, how well the se-

\* This work is supported by the National Bureau of Standards under contract #NB 79NAA B4369-1.

lected standard is supported by the present database machines and can be supported by the future machines. Third, the change of hardware architecture of a computing machine will have great effect on the design and implementation of a database management system. In particular, new hardware may change the interfaces among the components of a DBMS. Thus, the study of the standards for DBMS interfaces should take into consideration the present and expected progress in database machine research and development.

This paper reports on the results of a study conducted under the support of the National Bureau of Standards (contract #NB 79NAA B4369-1) to examine some of the proposed DBMS standards from the point of view of database machines. The emphasis is on the discussion of several issues related to data models and data languages and on how well they can be supported by database machines. The study aims to 1) assess the progress made in the database machine area, 2) determine the functional capabilities and limitations of the present database machines, 3) examine the issues on DBMS architecture, data models, and data languages from the point of view of present and future database machines, and 4) address some technical issues on the technology, the hardware, and software architectures of database machines.

## II. SOME LIMITATIONS OF CONVENTIONAL COMPUTERS FOR DATABASE APPLICATIONS

Several limitations found in the conventional computers motivate the study of database machines. They are:

### A. Mismatch of conventional computers for database applications

In 1948, von Neumann designed the programmable electronic digital computer for numeric applications. The design matched the technology of that day very closely to numeric applications. The semantic definition of numeric data was matched very closely to the storage representation:

<i>data semantics</i>	<i>random access storage</i>
$x, 26$	location 1, 26
$y, -5$	location 2, $-5$
$z, 1.7 \times 10^{23}$	location 3, $1.7 \times 10^{23}$

Using a random access storage, only a simple and efficient one-to-one mapping was necessary. Also, the semantics of numeric operations were matched very closely to the hardware instructions:

<i>semantics of operations</i>	<i>hardware instructions</i>
add	ADD
subtract	SUB
store in memory	STR

This close match allowed a very simple and efficient mapping. However, two significant things have happened since

that time. First, hardware technology has changed drastically. Cost and speed per function have improved by many orders of magnitude in the last 30 years. The rules of cost-effective packaging have changed from minimization of the number of logic gates and memory bits to minimization of the number of IC pins and packages. Secondly, the primary application for digital computers is shifting from numeric to non-numeric applications. In non-numeric applications, the user retrieves and manipulates data by specifying the attributes and values of the data he is interested in, i.e. addressing data by contents, rather than by addressing the memory locations where the interested data are stored. The basic operations required are Search, Retrieve, Update, Insert, Delete, Move-data, etc., rather than Add, Subtract, Shift, etc. The mismatch of the von Neumann design to non-numeric applications is the main cause of the complexity and inefficiencies of the present systems. The large and growing market for database systems warrants a reevaluation of the relationship between technology and database applications. If technology can be matched more closely to database applications, then perhaps advanced functionality, ease-of-use, and data independence can be achieved cost effectively.

In all considerations however, the most serious drawback is the lack of appropriateness of the sequential machine for the parallel process of data manipulation. One can liken this to the analogy of viewing a three-dimensional cube on a two-dimensional surface. Some forms are still recognizable; however, many others are skewed and hence do not appear 'normal.' So it is with processors. The software problems become necessarily more complex, simply because the representation is not appropriate. By providing a more appropriate environment, perhaps the 'skewedness' of present problems can be reduced.

### B. Many levels of mapping

Recent research efforts show that high-level data models and data languages which exhibit a high degree of data independence and ease of use are requirements to improve human productivity as well as act as logical interfaces with database systems. Currently, the implementation of high-level data languages and data models requires many levels of complex software to be executed, causing inefficiencies in system utilization and response. The software complexity and system inefficiency are due to the requirement that high-level commands and data views be translated into the low level machine codes and structures. In particular, software implementation of high-level data representations requires that auxiliary data structures such as inverted files, directories, and pointers, etc., be introduced to speed up data accesses for a particular set of applications. These auxiliary data structures must be properly maintained. This requirement complicates the updating operation, one of the most important database management functions, and significantly decreases its efficiency. Also, since these auxiliary data structures are tailored for a particular set of applications, a change in application often requires a large labor-intensive

software maintenance project. This considerably increases cost and time delays and decreases reliability.

### C. Performance bottlenecks

There seem to be two major performance bottlenecks in the present systems: the staging bottleneck and the communication bottleneck. In conventional systems, data are not stored at the place where they are processed. To "stage" data into main memory for processing is very time consuming, and often ties up the important resources of a computing system, such as communication channels. Database applications will continue to demand larger and more complicated databases, requiring more time to stage and process the data files. In order to support very large databases (greater than  $10^{10}$  bytes), or databases requiring both fast update and/or complex query, it is necessary to exploit specialized hardware to eliminate unnecessary data staging and to carry out database management functions efficiently. Data communication over long distances is expensive and limited in speed. This forces many database systems to physically distribute data to locations where usage is highest. Data redundancy is often purposefully introduced in distributed systems to avoid excess amounts of data transfer and to improve performance and reliability. However, many additional problems on data updating, recovery, integrity, and security in distributed systems are introduced by the above techniques. Special purpose hardware tailored toward managing distributed database management and supporting data communication would be very useful.

### D. User's increasing demands for DBMS capabilities

Database management system users are continuously demanding more sophisticated DBMS capabilities. Capabilities such as automatic database restructuring and system tuning, automatic data distribution and redistribution, backup and recovery, integrity and security controls, etc., are generally handled by software in the traditional systems. Tremendous overhead is generated in implementing these capabilities. Because systems are currently pushing software complexity barriers, performance improvements in this area are not likely without dedicating hardware to unburden saturated systems.

## III THE OBJECTIVES AND CHARACTERISTICS OF THE EXISTING DATABASE MACHINES

A database machine (DBM) can be defined as any hardware, software, and firmware complex dedicated and tailored to perform some or all of the functions of the database management portion of a computing system. The DBM may range from a small, personal query machine (intelligent terminal) to a large, public-utility information machine. We shall categorize the existing database machines into four categories based on their architectural distinctions and their

differences in objectives and characteristics. Each category of machine attempts to remove some or all of the limitations discussed in the preceding sections. In the following presentation, only the recent systems designed for general purpose database management applications are covered. Systems designed for text processing, document retrieval, sorting, etc., which are database machines in their own right, are not included.

### Category 1: cellular-logic systems

A cellular-logic system consists of a linear array of cells each of which contains a processor and memory element [47]. The general architecture of cellular-logic systems is illustrated in Figure 1. A database operation such as Search, Retrieve, Update, Delete, or Insert is broadcasted simultaneously to all the processors which carry out the operation against the data residing in their associated memory elements. Thus, in one rotation of the memory, the entire database is reached in  $1/n$ (th) of the time needed for a sequential search over  $n$  segments of data. Efficiency in data searches and other database operations is gained by the parallel processing elements. The memory elements of these devices can be disk tracks, bubble memories, CCD's, RAM's or other types of memories. The cells in these devices may communicate with their adjacent neighbors. This category of devices, thus, refers to a more general class of machines than the logic-per-track concept introduced by Slotnick [46].

The basic idea of cellular-logic systems is to move some of the frequent database management functions to intelligent secondary storage devices so that these functions can be carried out by the storage devices without the attention of the main processor. The data stored on the rotating devices such as disks, drums, CCD's, or magnetic bubble memories are systematically and exhaustively searched by the processing elements, one for each physical or electronic track of the rotating memory. Thus, data are processed on the same device where they are stored. Irrelevant data can be filtered out by the secondary storage devices and only the relevant data are brought into the main memory for further processing, thus avoiding the problem of staging described in the preceding section. Furthermore, since the entire database is exhaustively searched in each circulation of the memory, data can either be searched associatively by contents (i.e., by specifying what data are to be searched rather than where the data can be found) or by contexts (i.e., by specifying the neighborhood where relevant data can be found). The content and context search techniques in the cellular-logic devices offer uniformity and fast response time for search and update operations without the need to build and maintain special supportive structures such as indexes, hash tables, pointers, etc., used in the conventional systems. Data can be stored in these machines in a form very similar to the data structure defined in the conceptual schema of a database. Thus, the difference between the conceptual schema and the internal schema of a database in these machines is not as distinct as in conventional systems. The

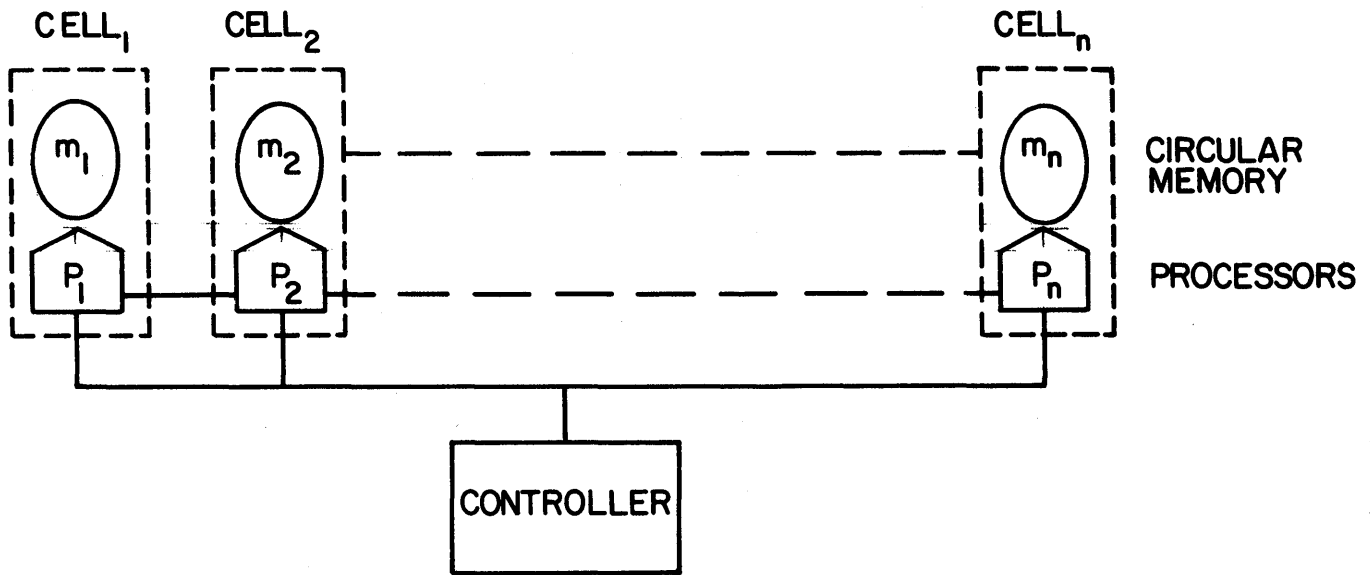


Figure 1—Cellular-logic configuration.

complex mapping between the two data representations can often be avoided.

Four basic architectural decisions which lead to an improved packaging of the technologies for the exhaustive associative search are examined as follows:

(A) The hardware consists of a regular arrangement of identical cells. The argument for this decision is as follows. First, the development and manufacturing costs of LSI and circuit boards are minimized, since only a single generic chip need be developed and manufactured and arranged uniformly on circuit boards. Second, reliability is improved because of the overall simplicity of this approach and because several simple schemes can be used to provide dynamic recovery from hardware failures. Third, the system can easily be expanded modularly without causing disruption to the system organization. As the database grows, increased storage is accompanied by increasing processing power, so response time remains independent of database size.

(B) Instead of using higher order arrays or tree structures, a one dimensional array of cells is used. The reasons are as follows. First, a one dimensional array minimizes the number of LSI pins per cell, since communication is restricted to fewer cells. Second, the number of pins per package is independent of the number of cells per package. This is very important, since it allows us to directly exploit the drastic, yet consistent, improvement in density, without increasing the number of pins per package. No other arrangement can accomplish this. Improved lithography and circuit designs promise to make further improvements by a factor of 100 in area density by 1990. Third, hardware utilization is most easily achieved using a one dimensional array, since fewer (only one) restraint must be met. For example, a two dimensional array requires two restraints. Users of the ILLIAC IV (Kuck [29]) have found this to be very awkward.

Furthermore, variable-length data objects can easily be linearized onto a one dimensional array.

(C) Each cell has a dedicated processor and memory. The reasons for taking this approach are as follows. First, experience has shown that using  $N$  processors that can access  $M$  memories leads to severe interconnection contention, so that neither processors nor memories are well utilized. A fixed one-to-one relationship between processors and memories allows an efficient utilization of both. Second, it also removes the complex reliability and packaging problems involved in a large interconnection switch. Third, the parallelism inherent in the exhaustive search can be directly exploited. Fourth, the amount of memory per processor can be varied to allow a family of database machines to be built using the same architecture. This allows trade-offs between cost and response time to be matched to different user environments and changing technology.

(D) Block-organized memories that are serially accessible within each block can be used, such as charge-coupled devices (CCD's), magnetic bubbles, and discs. These memories are generally cheaper per bit than memories that allow random addressing at the character level. Such memories will generally be classified as slow access. However, they are slow only when used to emulate a random access memory. When used for the exhaustive associative search, they are as efficient as a truly random access memory. In addition to searching efficiency, these devices offer efficient storage management for updates. Because of their dynamic nature, data can be inserted in place at the maximum data rate of the memory. Also, supportive data structures such as indexes, pointer, hash tables, etc., are eliminated and the effective cost per bit is further reduced. In summary, the block serial nature of these devices can be fully exploited to improve simplicity, efficiency, and data independence.

Several systems have been designed based on the cellular logic approach and some have gone through prototype implementation. A few of these systems are briefly mentioned here. More details can be found in the papers included in the special issues on database machines [9,24].

The CASSM project began at the University of Florida in 1972. The aim was to invest the hardware and software characteristics of various associative techniques. Direct hardware support of relations, hierarchies, networks, and string processing was investigated [11,32,48]. These hardware data types were implemented without any restrictions on length. Also, storage retrieval of instructions directly from the associative memory (associative programming) was studied. Associative programming is presently being studied at the University of Florida [50] under a continued NSF grant, with the CASSM architecture simulated in software [49].

The RAP project began at the University of Toronto in 1974. RAP [41,44] was intended to provide direct hardware support for the normalized relational model, with restrictions on the length of tuples. The RAP project also contributed to the understanding of several systems level considerations, such as the use of RAP as a staging device for very large file systems, and system throughput under a multi-user environment. Since its initial design, RAP has gone through some substantial changes. The most recent version, which is described in Schuster et al. [45], reduces the restrictions on the length of tuples.

The RARES project began at the University of Utah in 1976. RARES [30] provided hardware support for normal-

ized relations with length restrictions. The RARES storage structure was chosen to optimize output efficiency.

A research project called INDY began at Tektronix in 1977. INDY [10] directly implements a kernel language that is based on strings and classical sets with no hardware restrictions on length or cardinality. This kernel language acts as a meta-language that is generalized enough to directly describe various data languages and views, providing a simple closed mathematics for facilitating translations between views.

A recent project undertaken by Chang [7] at IBM, Yorktown Heights, investigates the use of magnetic bubble memories for supporting relational databases. A modular, configurable, electronically-timed magnetic bubble storage has been studied. The system follows the general concept of logic-per-track while a track in this case is a magnetic bubble chip with a modified major-minor loop organization. The proposed bubble chip configuration is shown in Figure 2. The storage minor loops are grouped to correspond to domains in a relation. The transfer line is segmented to allow the selection of a minor-loop group (i.e. a domain) to be accessed individually. The short buffer loops between the major and minor loops alleviate the problems arising from the rigid synchronization of the major and minor loops. The off-chip marker loops, being one-bit wide in contrast to being interspersed with many-bit large data records, can be quickly scanned to identify previously marked tuples. Since the minor loops allow parallel advance of data while the major loops only permit serial read-out of data, the quick scan fea-

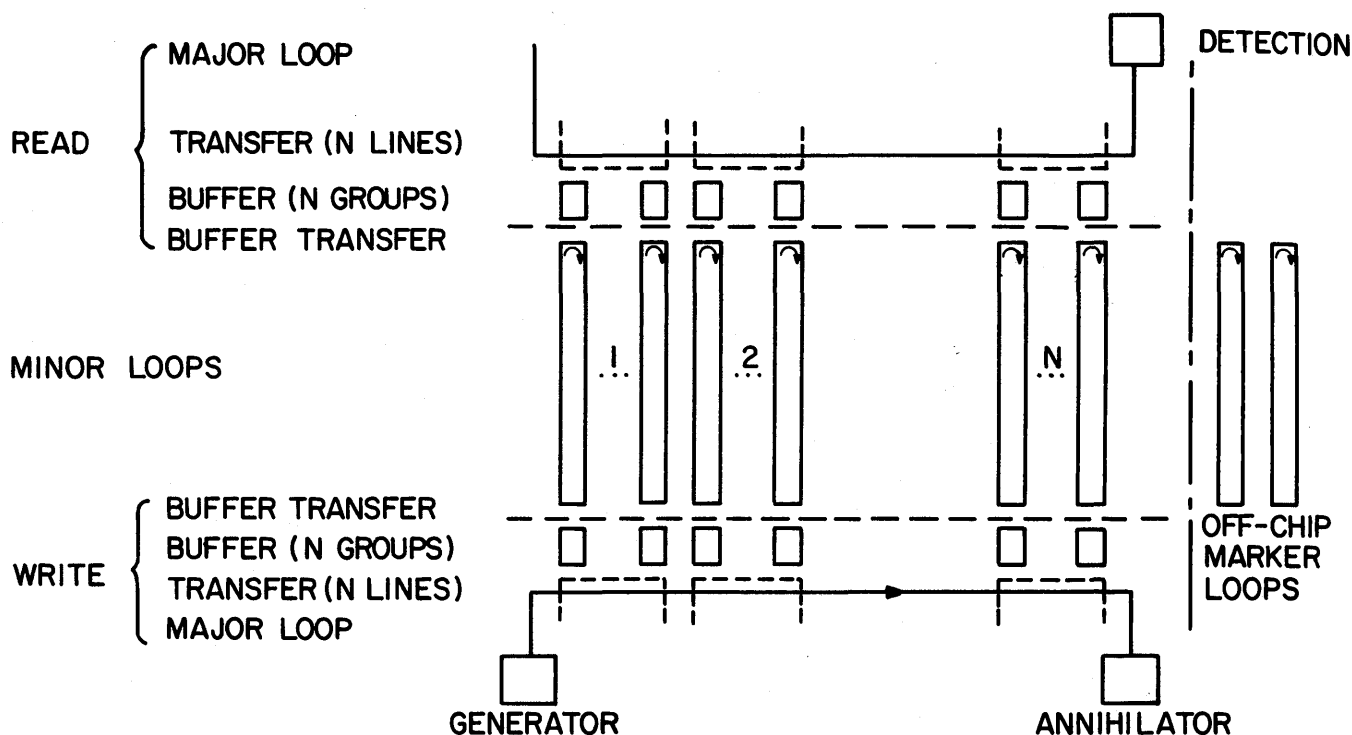


Figure 2—Modified major-minor loop organization.



ture of the marker loop can eliminate the output of unqualified data, thus greatly enhancing performance. The project clearly demonstrates that bubble memories have several desirable characteristics which can be utilized advantageously to support database management.

In summary, the distinguishing features of the cellular-logic approach are 1) increased processing capabilities in secondary storage devices to reduce the need for data staging in the main memory, 2) search time is independent of the database size, 3) elimination of the need for building, updating, protecting auxiliary structures, 4) the use of identical cells to increase reliability, flexibility in adding or reducing the number of cells and to reduce the cost of production, and 5) the potential for extremely high speeds as cell sizes decrease and memory density and speed increase (i.e. increase in the ratio of processing power to memory). Although most of the systems described here have gone through prototype implementation and testing, performance data from a real application environment is still lacking. The existing prototypes have rather limited processing capabilities. Many of the DBMS functions will still have to be handled by a conventional computer. Also, the staging problem described in Section II will not be totally eliminated if large databases are stored on archival memories and have to be moved to cellular-logic devices.

### Category 2: backend computers

Backend computers in database systems are dedicated computers for carrying out databases processing functions such as the retrieval and manipulation of databases, the verification of data access, the formulation of responses, the enforcement of integrity and security rules and constraints, etc. Backends are usually general purpose computers even though special purpose machines can very well be used. Figure 3 shows one possible configuration, the operating system, application programs, and DBMS interface run on the host computer, and the actual DBMS runs on the backend computer.

The key concept of backends is to off-load the database management functions from the host computer to dedicated processor(s) in order to 1) release the host from tedious and time-consuming operations involved in database manipulation, maintenance and control, and 2) increase system performance through functional specialization of and through parallel processing among the host and the backend(s). The primary impetus for the backend approach is, of course, to reduce the cost of managing data. The backend approach can be viewed as a cost-effective alternative to upgrading the host or to achieve the level of functionality and performance that no conventional system can provide.

The isolation of the DBMS, the mass storage devices and the database from the host can bring a number of additional advantages. First, several hosts, possibly dissimilar, can share on-line data in the configuration shown in Figure 4. A single backend may handle the processing of the database and present data in forms suitable to the dissimilar hosts. Second, databases and the DBMS itself can be transported

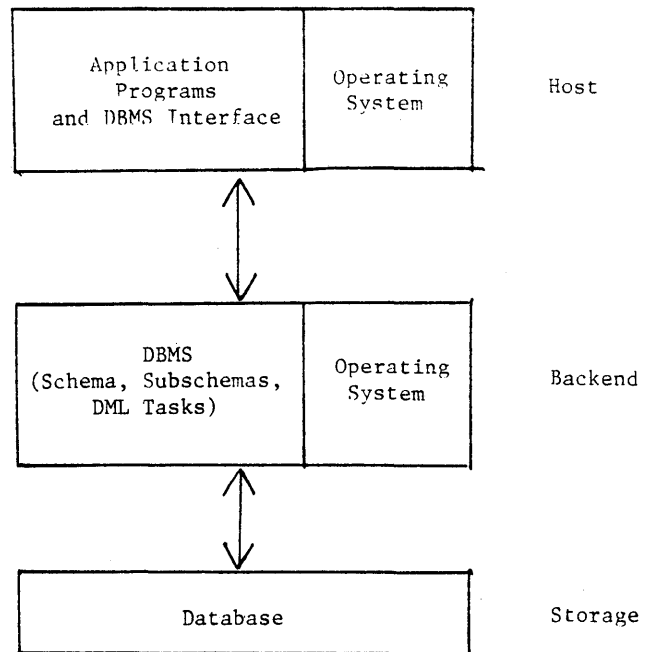


Figure 3—A configuration of a backend computer system.

from an old mainframe to a new one with relatively little conversion effort. Similarly, changes to the databases, the mass storage devices, and the DBMS (e.g. adopting a standard DBMS) can be made without entailing changes to the host. Third, storage devices including special purpose cellular-logic devices or bubble devices can be made available through backends to mainframes that do not otherwise support these devices because of I/O or operating system constraints. Fourth, multiple numbers of backends (see Figure 5) can be used to process large databases which can be stored either in a distributed manner across secondary memory devices to facilitate parallel processing or in a manner such

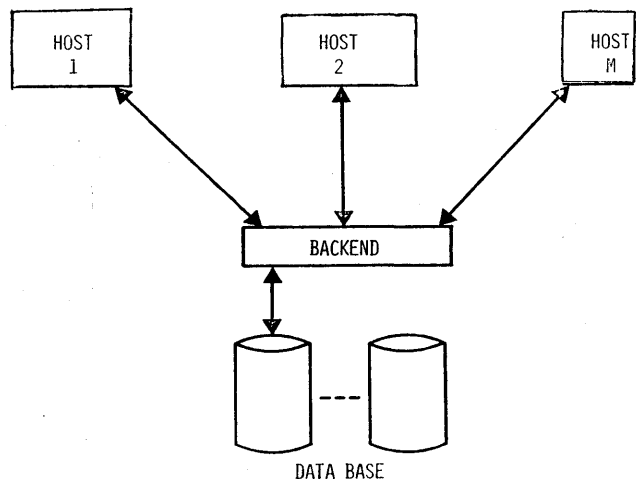


Figure 4—Multiple host configuration.

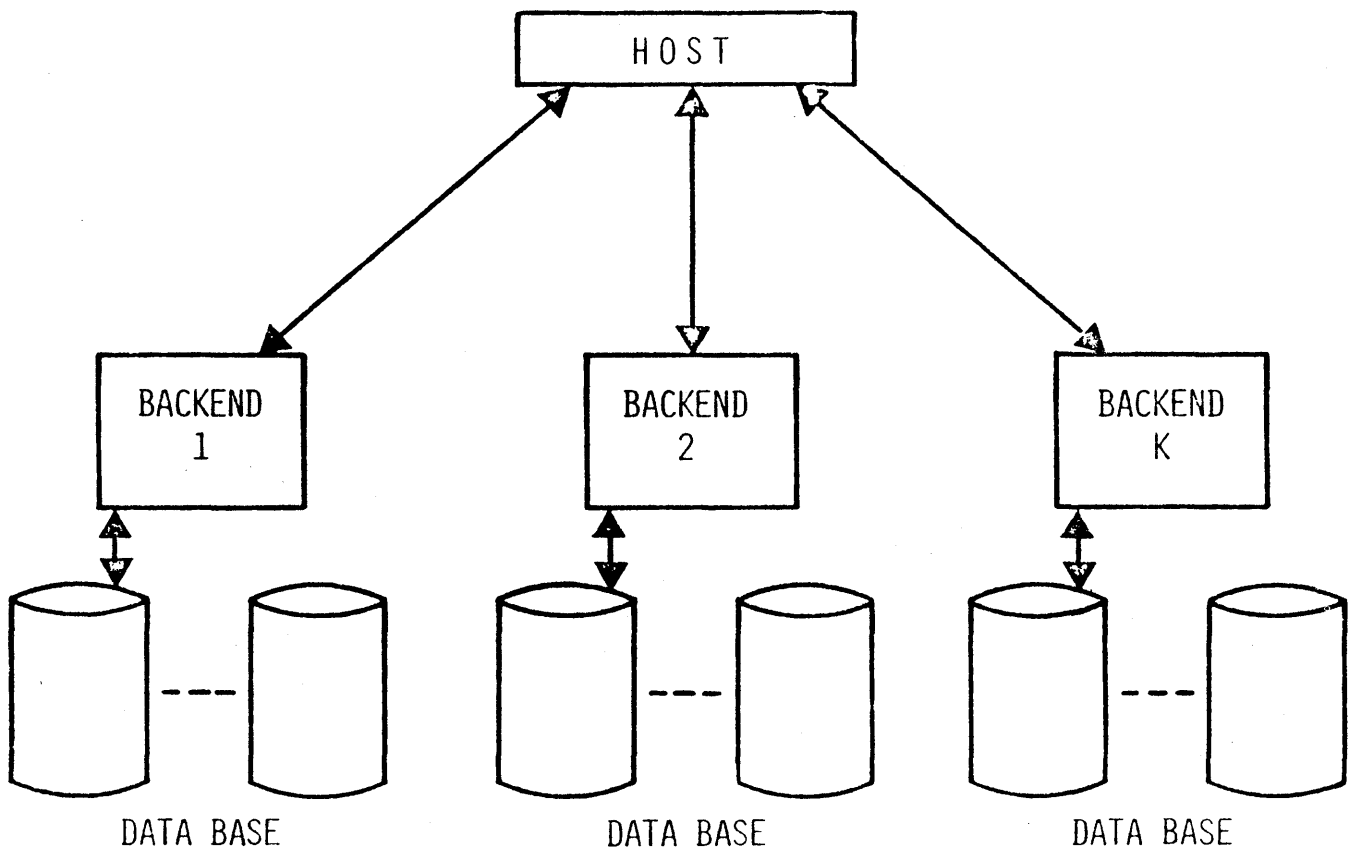


Figure 5—Multiple backend configuration.

that one database can be processed by one backend. Lastly, the enforcement of database integrity and security can be separated from that of operating system integrity and security; thus the failure of one will not endanger the other.

The first development of the backend system occurred at Bell Laboratories [5]. This system was called the Experimental Data Management System (XDMS) and was undertaken to both demonstrate the capability of the backend concept as well as implement the new CODASYL DBMS specifications. The implementation required eighteen months and six man years of effort. The system was implemented to a level of experimental usefulness and the concept was verified.

The Data Computer is another example of the backend processor approach. It is a large-scale database management system running on a PDP-10 and has been implemented for use in ARPANET [36] by Computer Corp. of America. The Data Computer essentially provides facilities for data sharing of a single database among dissimilar host computers in a network environment. That is, it is implemented through a communication scheme involving the identification of the host processor type so that data to be retrieved and sent by the Datacomputer can appear in the format expected by the requesting host. Likewise data which are to be stored by the Datacomputer are converted upon receipt from the identified

host and stored for use as the originator sees it. With such a scheme, the amount of storage can be continually expanded, performance can be maintained by replicating the systems, and the backend machines are available to all hosts in the network.

Some additional developments indicate the possible direction in which this movement may be heading. In the past few months, Cullinane Corporation made available to four government agencies IDMS implemented on a PDP 11/70 capable of supporting an IBM or IBM-compatible host. One participating group (within the Navy) is just now beginning a very serious evaluation of the utility of such a system in their production environments to extend the useful life of their existing computing facilities.

During the period of time Cullinane Corporation was implementing IDMS for use in a backend, Kansas State University [16,17,37], under a grant from the U.S. Army Computer Systems Command, was developing a prototype network system built around a machine independent, high-speed bus system (20 mega bytes/sec transfer rate) which would permit heterogeneous computers to communicate in any topology desired. With this communications support software finished, a natural application was the backend environment. The software design documents were furnished to Cullinane along with the host software. Addition-

ally, Cincom's DBMS system (TOTAL) was modified to run on an Interdata 8/32 backend from either the IBM host or another mini in the network acting as a host.

A great deal of database machine activity is occurring in Japan. One project defines a database machine called GDS—a generalized database subsystem—which has a sufficiently low-level interface to provide potential support for any data model [18]. One major contribution is its ability to interface directly to the main memory of its host so that I/O overhead incurred by the host CPU during large data transfers can be avoided.

The existing backend systems are still experimental in nature. The desirability of backend is yet to be proven by performance evaluation and measurement of "real" systems. In conclusion, the idea of extending the functionality and performance of a mainframe by dedicated backends is a sound one. However, this approach does have its adverse problems. For example, the backend(s) introduces different hardware with the attendant problems of maintenance, software support, and the additional procurement effort and cost. Also, the balanced assignment of DBMS tasks to the host and the backend(s) is not a simple problem. More discussions on backends can be found in [33,42].

### *Category 3: integrated database machines*

This category of systems uses a number of functionally specialized processors, which can be general-purpose and/or special-purpose processors, to implement the processes of a DBMS. Systems of this type may use, for example, specialized associative processors for the processing of directories and mapping data, intelligently controlled disks and mass storage devices for the storage and processing of the major portions of the database, a system processor for general coordination, and dedicated hardware for security control. By the use of the functionally specialized hardware and the parallel processing capabilities of a family of machines, these systems aim to achieve greater efficiencies in database management. The highly modular family of machines gives users the opportunities to mix and match process and storage capacity.

Different from the cellular-logic systems in category 1, this category of systems are larger and more complete systems of which a category 1 system can be a component. The specialized hardware units used in these systems are quite different. They lack the uniformity of the cells in category 1 systems. This category also differs from category 2 systems in that functionality and performance are achieved mainly by hardware (and thus software) specializations rather than software specialization alone used in the existing backends. It should be noted, however, that the distinction made would not be clear if special-purpose hardware devices were used in the backend systems. Nevertheless, we can say that the design of this category of systems involves treating hardware, software, DBMS, and databases as a whole rather than simply extending the capability of a given mainframe using backends.

Some example systems of this category are the following. The Data Base Computer (DBC) project at Ohio State Uni-

versity proposes an architecture where every major DBMS function has a dedicated processor and whose overall organization exploits pipe-line parallelism [1,3,20,21,22,23]. It contains various associative processors for logical data model and disk memory mapping. It also proposes several architectural changes to moving head disks to increase bandwidth an order of magnitude over today's secondary storage data rates. The integration of the security function into the DBC's architecture is also considered.

The RAP.2 effort at the University of Toronto has expanded its research by formulating the RAP (a category 1 machine by itself) associative processor's role in an integrated database machine. Most of the work has centered around data partitioning or staging strategies where database and schema data reside partially on disk and partially on associative processors [45].

The INFOPLEX system proposed at MIT is an example of integrated database machine architecture [35]. It utilizes new microprocessor capabilities by organizing a memory and processor hierarchy which takes advantage of the parallelism inherent in concurrent requests to maximize performance.

Another direction is to make use of low cost currently available microprocessors to form a simple network system for processing distributed databases using a single-instruction multiple-data stream architecture (SIMD). In this case, segments of data files are stored across memory devices each of which is dedicated to a microprocessor. Software tasks for a database management system are simultaneously carried out by the processors against the contents of the local memory. This alleviates much of the switching time overhead found in a network systems with shared memory. A recent example of this approach is the MICRONET system being developed at the University of Florida [51] using a PDP 11-60 and four LSI-11 computers.

Another multiprocessor system called DIRECT [15] is designed for supporting relational database management systems using a multiple-instruction multiple-data stream (MIMD) architecture. Microprocessors are dynamically assigned to a query depending on its priority and the type of relational algebraic operators it contains and the size of relations referenced. The system is being implemented using LSI-11/03 microprocessors and CCD memories which have associative search capability.

In summary, the main characteristics of this category of database machines are 1) the use of functionally specialized hardware to achieve efficiency, 2) the system's approach to the design of hardware, software, DBMS, and databases, and 3) the modular family of machines allows users to exploit parallel processing and pipelining techniques. However, the hardware interconnection, the data and program communication, and the operating system support in a system using dissimilar hardwares can be rather complex. The proper identification of DBMS functions for implementation in hardware remains a challenge.

### *Category 4: high-speed associative memory systems*

In this category of machines, a high-speed associative memory is used together with conventional memory devices

such as core memories, rotating memories, or shift registers to form a hierarchy of memories for data processing. Databases are stored on conventional secondary storage devices. Data are moved from the slower secondary storage to the associative memory for high-speed searches by content or context. The same characteristics which make a cache for speeding up data reference of main memory are used here to improve data access to secondary storage. Figure 6 shows a typical configuration of this type of system. The associative memories used in these systems differ from the cellular-logic systems in that each bit or each word rather than a segment of memory has a processing element. Associative searches can be carried out in all bits or words of the memory simultaneously and thus are much faster than the sequential scan of memory segments in rotational devices. The technology used for high-speed associative memories is faster than the rotation devices. However, it is far more costly.

A good example of the high-speed associative memory approach is the STARAN computer system [2,12,43]. The key element of the system is a set up-to-32 associative processor arrays which provide content addressing and parallel processing capabilities. Each processor array is a multi-dimensional access memory matrix containing 256 words by 256 bits with parallel access to a maximum of 256 bits at a time. The access can be in either the word or bit direction. Associated with each word of a processor array is a processing element which examines the content of the word and manipulates the word bit-by-bit serially. Control signals are broadcast to the processor elements in parallel by the control logic unit and the processor elements execute instructions simultaneously. Data stored in the main or secondary storage of a conventional computer system are paged in and out of the processor arrays for associative searches.

Program instructions of the associative processor are stored in a control memory which consists of three fast page memories made of volatile, bipolar, semiconductor elements and a core memory block. Program segments stored in the core memory block are paged to the fast memories before execution. The control logic unit fetches and interprets the instruction from the control memory and transfers control signals to the processing elements of the processor arrays to manipulate data in the arrays.

Although the associative array processor was originally built for air traffic control and other real time sensor surveillance and control applications, the content addressability and parallel processing capabilities of the processor provide many desirable features for database management. A DBMS built around a four-array STARAN has been reported by Moulder [40]. Other work based on this system and a hypothetical associative memory for use in a database management environment can be seen in DeFiore and Berra [13,14], Berra and Oliver [4], and Linde et al. [31].

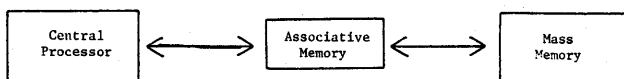


Figure 6—A typical associative memory system configuration.

The principal benefit of this approach is improved performance. The use of high-speed associative memory reduces the effective access time of the mass memory where databases are stored. However, due to the high cost of building this type of memory and processor, the size of associative memory is rather small. In a database management environment, considerable amounts of data will have to be paged in and out of the associative memory to take advantage of its capability. Although data can be searched in high speed once the data are in the memory, to stage data into the memory can become a bottleneck of this type of system. For certain types of applications such as table look-up and directory processing, the use of high-speed associative memory will result in an order of magnitude improvement in performance at relatively low incremental cost. Where there is little locality of references, however, the potential cost benefit will not be realized.

#### IV. DATABASE MACHINES AND SOME ISSUES ON DBMS ARCHITECTURE, DATA MODEL AND DATA LANGUAGE DESIGNS RELATED TO DBMS STANDARDS

Having described the motivation, objectives, functionalities, and challenges of the existing database machines, we shall now look into some of the issues on DBMS architecture, data model, and data language design from the viewpoint of database machines. Many issues discussed here have often been raised by researchers and practitioners. They are very relevant to the standardization of DBMS architectures, data models, and data languages.

##### A. DBMS architecture issues

###### DBM's support of multi-schema architectures

The DBM technology could conceivably make those DBMS architectures which involve multiple numbers of schemas (e.g. the ANSI/SPARC architecture) very cost-effective. That is, it could have performance features that reduce the cost and complexity of the various schema mappings. The commitment to separate user views, logical data structure, and physical data structure stands on its own right. It is not compromised by the fact that we are limited to von Neumann processors, disk, tapes, etc., today and it should not be compromised by what happens tomorrow, particularly since we can make the separation increasingly economical through DBM technology. With respect to the standardization of the DBMS architecture, it cannot be stated categorically that DBM technology as such is going to push us toward a particular conceptual data model and external data models. Rather the DBM will probably support whatever is wanted as the "best" conceptual data model (by whatever criteria) and its mappings to external models and the mapping to the internal data model (including the internal data model of the DBM itself, the distribution among various mass storage devices, and distribution among geographically separated database systems). The internal data model is

probably not "standardizable," because, first, it does not need to be. Programs and end users do not see it or depend on it. Secondly, it must adapt to changing storage technologies including the DBM, storage hierarchies, geographically distributed databases, etc. Therefore it is important to separate the internal schema from the conceptual schema and keep it flexible and extensible.

#### **Mappings between external and conceptual schemas**

The mapping between external and conceptual schemas may involve a subset mapping and a restructuring mapping. Subset mappings are necessary to provide privacy from unwanted queries, security from unwanted updating, and user convenience by removing all data that are not of concern to the user. Restructuring mappings are necessary to provide data structures that are convenient for user applications, and to provide support of multiple user models and languages.

A DBM can play an important role in implementing these mappings with efficiency and simplicity. It is possible to store and manipulate these schema descriptions on database machines as simply another database, where mappings are accomplished using queries to the schema descriptions. However, to do this, database machines must be capable of a more generalized pattern matching capability for strings and sets. This is necessary since these schema descriptions usually involve searching abstract or axiomatic (e.g., set theoretic or predicate calculus) representations, rather than simply searching actual data instances. Ideally, the same hardware would be used for actual data and for both external and conceptual schema descriptions.

#### **Mappings between conceptual and internal schemas**

Some database machines can allow the storage structure of a database as defined by the internal model to be very similar to the structure defined in the conceptual model, and thus simplify the mapping process. For example, a relation in the community view can be stored and searched in an associative memory without index tables, hash tables, pointer arrays, etc., commonly introduced in conventional systems. This means that any data stored on these machines requires only the simplest of mappings to its internal schema. However, this does not necessarily mean that the internal schema of the entire database system will be simpler. In a large database system, an associative memory would probably be one out of a whole hierarchy of memory devices, each featuring its own tradeoff between cost per bit and response time. If the associative memory is used and managed as yet another component in a large system, it could add some complexity to the overall internal schema. Instead, the architecture of the entire database system should be reexamined with database machines in mind. Its unique qualities can be exploited to simplify the overall system. The unique features of associative machines are fast response times and simple mapping between the conceptual and the internal schemas but with a higher cost per bit than mass storage devices. The following three systems functions seem appropriate for associative machines.

One function is the direct storage of databases whose requirements for speed warrant a higher cost per bit. A second function is to manage the mappings between the conceptual and internal schemas for databases stored on mass storage devices or for geographically distributed databases. The distribution of data among various mass storage devices or among geographically separated systems can be described and stored directly in associative machines as simply another database. Schema mappings can be implemented using queries from the internal and conceptual schema descriptions. Associative machines offer the potential for storage and querying of abstract representations. An internal schema that uses abstract representations, rather than involving actual data instances, has the potential advantages of a more compact description and one that requires no updating when updates are made to the actual data.

A third function of associative machines is to act as a staging device for large blocks of mass storage. Most mass storage devices are accessed by location. Efficient use of these devices usually requires clustering of data into many large physical blocks, which is biased to certain access paths. After queries to the internal schema (directories) have reduced the number of blocks involved in a retrieval to a small number, associative machines can then be used to further search these blocks.

#### *B. Data model issues*

##### **Database machines support of data models**

A DBM can be implemented to support any existing data model. For example, RAP, RARES, and DIRECT were designed specifically to support the relational model. The CASSM and INDY systems can support hierarchies as well as a subset of relational algebra operators and string pattern searches. The ASP system was designed to support a form of the network model. Although it was not compatible with the DBTG model, such an implementation should not present any major problems. Finally, any general purpose back-end computer can be programmed to support any or all of the models simultaneously.

The implementation in hardware of a single model does not preclude it being used to support other models. For example, a system that directly supports relations can be used to simulate hierarchical and network models. They can be implemented by setting aside items called "associative links" or "context pointers," in record occurrences (tuples) to store identification and structural data.

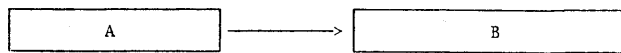
Implementing hierarchies and networks requires the ability to implement "functional associations" between occurrences of record-types [52]. A record-type is analogous to a relation. A functional association can be defined as a 1:N (i.e., a one-to-many) linkage or mapping between record occurrences of two relations. That is, if a 1:N linkage exists between relations *A* and *B*, then one record occurrence of *A* can be associated or linked with zero or more unique records of *B*. Each *B* record will have at most one *A* record for a particular association. An association or link is equivalent to a "set" in DBTG terminology. Restrictions on the ap-

plication of functional associations between record-types determine if the database schema is hierarchical or network.

One way to implement an association is to allocate an item called ASSOC in the relation that acts as the domain of the functional association. This scheme is shown in Figure 7. The item ASSOC acts as the associative link. Each record occurrence must have one item whose value uniquely identifies the relation and each particular occurrence within the relation. This item will be called ID for identification. For each record of *B* that is associated with one record of *A*, the record ID value of *A* is stored in the ASSOC item of *B*. Finding records of *B* associated with a particular record of *A* or vice versa is simply a matter of using the associative cross selection or join instructions which interrelate two relations through comparable ID and ASSOC values.

A second way to associate records of the same or of different types is to create a new linking relation which contains two (or more) ID items—one for each record-type. This relation, called LINK, associates one record of *A* to one record of *B* by storing the associated ID's of the two records in one occurrence of LINK. This scheme has the advantage of implementing *M:N*, or many to many, associations between record-types. An example is shown in Figure 8.

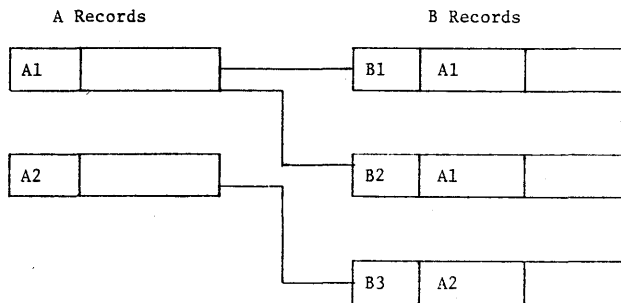
It should be noted that only "information carrying" associations need be implemented with links. All other relationships which can be derived directly from the values in



a) 1:N association between record-types A and B.

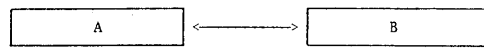


b) Record-types with associative link fields.

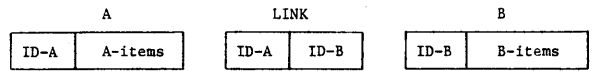


c) Example record occurrences.

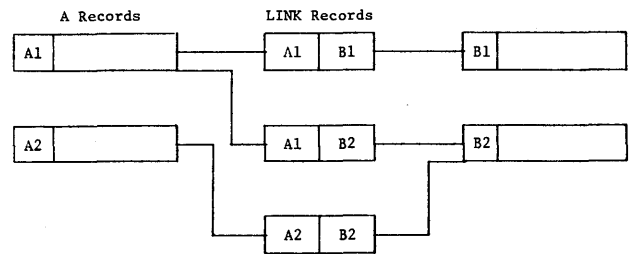
Figure 7—Implementing associations with relations; a) 1:N association between record-types A and B, b) Record-types with associative link fields, c) Example record occurrences.



a) M:N association between record-types A and B.



b) A and B record-types with LINK relation.



c) Example record occurrences.

Figure 8—Implementing associations with a LINK relation; a) *M:N* association between record-types A and B, b) A and B record-types with LINK relation, c) Example record occurrences.

the records can be handled directly through associative cross selection or join instructions of relational DBM's.

Of the three data models, the relational model is the most general in terms of the types of associations it can represent. It also requires the least number of basic or primitive operations to implement a relationally complete data manipulation instruction set. Also, its simplistic record structure and orientation to sets-of-records operations makes it a natural candidate for DBM implementation.

From the above comments, it may appear that the relational model may be the easiest to implement and result in the best performance. However, we must be careful about jumping to conclusions. Many of the additional features of hierarchical and network models were proposed because of the need to improve transaction processing performance. The same techniques that have served software implementations will likely serve hardware as well. Also, the users application may better lend itself to hierarchical or network modeling. In such cases, hierarchical or network hardware will probably out-perform relation hardware using software and data to simulate other models' primitives. Also, many transaction applications do not require complex search nor are the sets of records to be processed large. In fact, today's online transaction processing applications are dominated by having a large number of concurrent transactions requiring relatively simple search and update interactions. These types of operations are the least likely to take advantage of the set-oriented associative processing capabilities of relational or set theoretic DBM's. Of course, a major reason why existing computerized database applications predominately require simple searches and updates is that an adequate implementation of more complex models is not available.

Judging from existing examples, the DBM will very likely make the more advanced conceptual data models (e.g. relational or set-theoretic) more feasible to implement, where as today they are frequently judged very complex to implement efficiently as a general purpose system for a broad base of applications. Thus we should be able to choose a standard model based on user benefits and assume with confidence that the performance gap will gradually close.

### C. Data language issues

We now turn to data languages, collectively consisting of all languages for directly manipulating database data on behalf of application programs or end-users. Thus data languages include data sublanguages, which are extensions to conventional programming languages, and self-contained languages (such as query languages, report generators, "query by example" and other end-user interfaces). Data sublanguages in particular are the target of standards efforts because of the need to protect the user community's investment in computer programs that use these interfaces. Any practical standard takes into consideration user requirements; e.g., proper functionality and ease of use, and feasibility—is there a reasonably efficient, economical implementation of the proposed interface? The feasibility condition creates tension in times of rapid technological innovation, when ground rules for judging what is possible or economical are subject to radical change. This appears to be the case for data languages, not only because of DBM development, but also in view of the slow but steady trend toward hierarchies of storage and geographically distributed data processing. The following paragraphs tell this story: The bad news is that the ability to improve price/performance through technology is very sensitive to the character of the data language. The good news is that we can predict well in advance what features data languages must have to fully exploit emerging technology. Furthermore there is a strong indication that these same features are desired by the user community independent of technology considerations. If so, then the standards makers have their work cut out for them.

#### High level vs. navigational data languages

As will be seen, the underlying technical considerations generally motivate the development of very high level data languages, by which we mean languages in which the user/programmer expresses to the database system *what* results are expected instead of, or in addition to, *how* the results are to be obtained. With regard to high level data languages it must be recognized that:

- "high level" and "low level," like "procedural" and "non-procedural," are relative terms;
- self-contained languages are not the only languages that can be high level or non-procedural. There is no intrinsic reason why a data sublanguage cannot be high level even if the programming language in which it is embedded is low level. See, for example, the use of ALPHA in [8].

Whereas everyone appears to agree that end-user oriented languages should be high level, there is an ongoing controversy concerning whether high level data sublanguages are desirable. On the one side are those who argue that programmers should have relatively low level facilities so that they can fine-tune performance tradeoffs. The other side contends that in an era of increasing programming costs and decreasing hardware costs it is best to optimize programmer productivity through the use of high level facilities and let the system worry about efficient hardware utilization. Technology trends and the DBM in particular strongly support the latter position. We will briefly examine some reasons for this.

A database machine can sometimes be "tightly coupled" to the hardware which makes use of it. For instance a mainframe manufacturer could develop a backend which is enclosed within the host itself and communicates with main memory through a very high speed bus. Or a multifunction terminal might be plugged directly into a small "query machine." In such cases there is no concern that communication with the DBM will be a performance bottleneck. But suppose that the DBM is not developed by the host manufacturer, or that is designed to serve multiple hosts. Or suppose that a DBM is required to communicate with remote hosts in a network, or even with other DBMs to support a distributed data base (Figure 9). The need to do all of these is bound to arise, so the DBM developer must evaluate the response and throughput implications of loosely coupling the DBM to an external I/O interface or an even slower telecommunications channel. There is nothing in the ten-

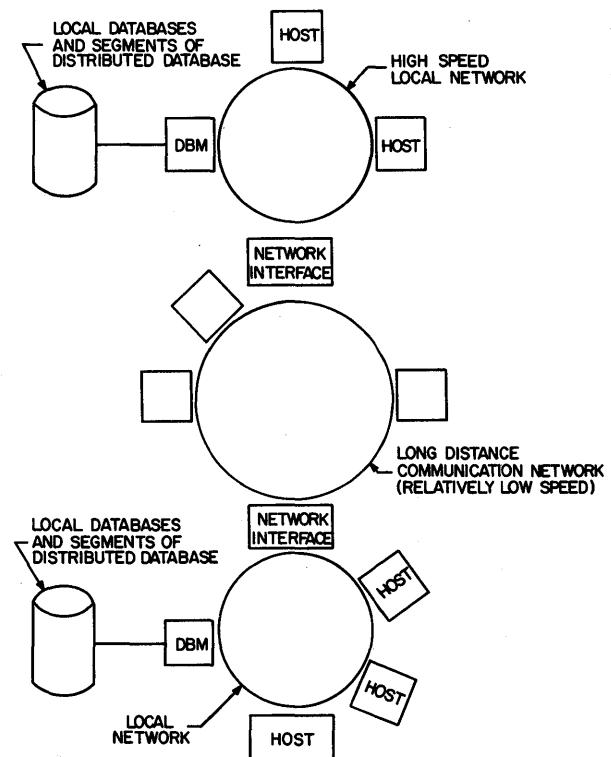


Figure 9—DBM nodes in a distributed environment.

year picture to suggest that the price/performance penalty for loose-coupling will go away (otherwise the economic argument for distributed data processing would lose most of its force).

The DBM developer is therefore motivated to minimize the amount of data that must go in or out of the DBM in order to get a user's job done. He must also strive to minimize the number of separate messages, large or small, to reduce the communication burden. All of this has a direct bearing on the data language available to the user. In the extreme case, if the user can express his job in a single data language statement, and if that statement can be directly interpreted by a DBM, then obviously the communication overhead has been reduced as much as is possible. If, in contrast, the job *must* be decomposed by the user into a program with several lower level sublanguage statements, possibly executed in a loop, then the number of messages and amount of data transferred will increase dramatically. For example, suppose the user is to mark "inactive" all posted accounts for which there have been no debits or credits during the last twelve months. Given a powerful data language capable of dealing with entire sets of data, this transaction can be expressed with a single statement—a single "call" to the database system and no database records transferred. Given a record-at-a-time ("navigational") data language, there would be at least two calls to the system for each inactive account, one to retrieve the record and the other to store the modified version.

There are halfway measures which preserve the navigational nature of low level data languages, but would still reduce some of the DBM interaction. For instance, high level *intention declarations* are a possibility (Lowenthal [34]). If, in the above example the user could state in some fashion, "I intend to update all accounts for which there have been no debits or credits posted during the last twelve months," then the system could subsequently buffer blocks of multiple records between the host and DBM, but move one record at a time to or from the user's program. This wouldn't reduce the amount of database data transferred, but it would cut down the number of messages between the host and DBM (each message would be longer). This technique is useful when sequential treatment of data is ultimately unavoidable by any means, such as if a program is required to produce a list of the accounts that have been marked inactive.

Consider another method of capturing the high level meaning of an operation expressed in a low level data language. Suppose that the results to be obtained are such that the programmer can write a special kind of subroutine in which the only data referred to are the parameters, the database data retrieved in the subroutine, and some constants established in the sub-routine. This subroutine does not refer to global (common) data, does not read or write non-database files, and does not call other subroutines. Given such constraints, it is feasible to transfer the entire subroutine to the DBM as a single operation, either in source or object form. The DBM can perform internal retrievals, returning only the subroutine's output to the host. Using the above example, a subroutine *X* would be catalogued (in the DBM) which retrieves each qualifying account and stores it back with the

"inactive" indicator set. The only interaction between the host and the DBM is the command to execute *X* and status returned upon completion. An additional benefit of this approach is the opportunity for the DBM to optimize the execution of *X* since it "sees" the entire collection of database operations instead of individual data language statements. CASSM is an example of a DBM which supports catalogued subroutines.

There are several data language features that could be included if the aim were to minimize communication. Most of these motivate or force the user to express at a high level what is to be ultimately accomplished. They cause the language to be less procedural, or supplement procedural sequences with non-procedural declarations.

We point out in passing that the cost of inter-task communication in a typical mainframe operating system is surprisingly high, so that even in a conventional software database environment there is a strong motivation to reduce the traffic between the application task and the database task. Another independent motive is fueled by the advent of hierarchies of storage, which are inevitable if very large databases are to be addressed in the context of foreseeable price/performance trends for different types of secondary storage; no single device is expected to emerge both cheaper and faster than any other device (see Figure 10). It has been argued that storage hierarchies will be more effective if the data staging algorithm can anticipate in advance exactly what data will be required [34]. This again relies on a language through which the user can express with some refinement his data needs. High level, set-oriented statements, intention declarations and the like would all marry quite well with an intelligent data staging mechanism. Thus it is the broad direction of computer technology encompassing distributed processing, storage hierarchies, and software engineering, and not just the DBM which calls for a reassessment of data language standards efforts.

#### Set oriented vs. record oriented processing

The DBM concept most directly and vividly exposes the relationship between a data language and the hardware mechanism which ultimately does the work. In previous sections it has been established that conventional computer architecture is not particularly well suited for database management, that dramatic improvements in cost/performance can be achieved with fundamentally new approaches. In nearly every proposed architecture, be it oriented to searching, sorting, list merging or the like, there is a common theme: one or more sets of data are operated upon to produce another set. This is no accident since the basis for the claimed economy is parallel processing, that is, many small inexpensive processors working effectively together to do a large job quickly. The opportunity to exploit parallelism practically depends on the ability to define operations in terms of sets instead of individual points of data. This in turn clearly depends on the ability to deal with sets of data at the level of the data languages itself.

In the world of scientific computing, scalar oriented lan-



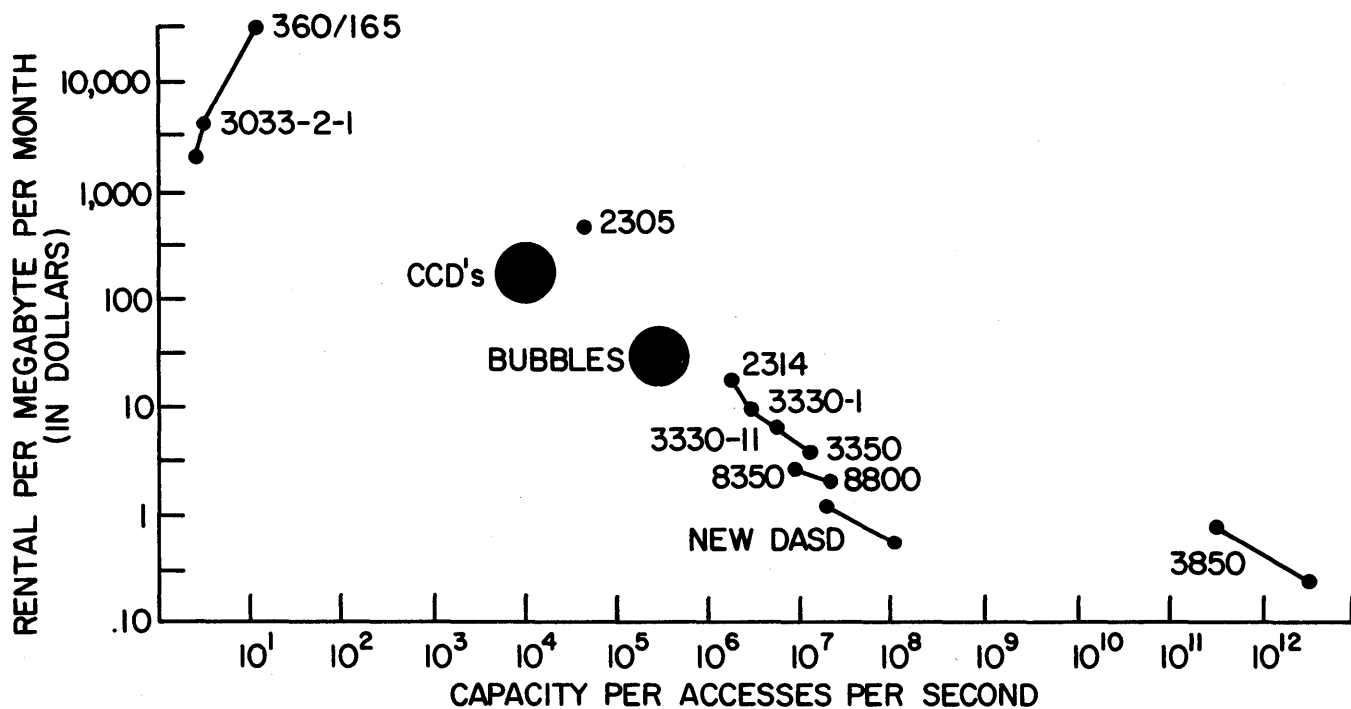


Figure 10—Trends in online storage—future product directions.

languages like FORTRAN have been enhanced with high level array operations so that, for example, matrix inversion or multiplication can be expressed as a single statement. This enhancement is motivated not so much by software engineering principles as the industry's ability to build highly parallel machines that operate on arrays at blinding speeds. If matrix multiplication can only be expressed as a sequence of DO, IF and assignment statements, how can the underlying system figure out what the programmer intended? How can the advanced architecture be exploited? Likewise if a database programmer cannot express a predicate as a predicate ("find all accounts for which no credits or debits have been posted during the last 12 months" paraphrased as a single data language statement), but must restate it procedurally with more primitive record oriented statements embedded in loops, how can set oriented DBM's like RAP, RARES, or CASSM be effectively exploited?

In the past, set oriented data languages, sometimes (incorrectly) called "relational" languages, have been regarded as powerful but impractical—too expensive to implement and operate. The lower level record oriented languages, including the CODASYL DML, have scored high points for feasibility and economy. The emergence of DBM technology may actually reverse this situation in the next few years. In view of this, language developers working with the CODASYL basis should work out ways of enhancing the DML with set oriented operations. Not only will this result in a better fit with the DBM, but also with the trends in user requirements (people productivity), mass storage technology and distributed databases.

There is an obvious counterargument. If users rarely need to manipulate matrices, then fancy scientific computers

should be built for the few and FORTRAN for the masses shouldn't be affected. Likewise if very few users need to manipulate sets of data, but rely mainly on sequential access or simple direct access ("find the unique account record with key account number 745286"), then set oriented machines will not have broad appeal. We strongly believe that although there will always be a need for record oriented access to data, there is also a great demand for set oriented capabilities. Moreover this demand can only increase as databases come to be regarded as information resources for management.

## V. SOME TECHNICAL ISSUES ON DATABASE MACHINES

The following is a collection of key technical issues which must be addressed by researchers in database machine technology. The discussion is broadly grouped into three areas: basic technology, hardware architecture, and software architecture.

### *Basic technology*

The use of the systems described in Section III will depend heavily on cost, performance, storage capacity, and reliability of such solid-state devices as LSI processors, RAMs, CCDs, and bubbles. DBM architects will be structuring systems which incorporate such large volumes of these devices that reliability will dominate the design of products. Researchers are only beginning to realize that solid

state devices are not just "electronic" disks. Bubbles and CCDs provide unique opportunities for combining logic with storage as demonstrated in IBM's bubble query machine, RAP.2, etc. The main manufacturing problems for research and development are:

### 1) High density storage media

Texas Instruments introduced in 1976 the TIB 0101 bubble chips with  $10^5$  bits/chip at  $10^8$  bits/in<sup>2</sup> density (6  $\mu$ m bubble diameter), and in 1978 the TIB 0103 bubble chips with  $2.56 \times 10^5$  bits/chip at  $4 \times 10^6$  bits/in<sup>2</sup> density (3  $\mu$ m bubble diameter). A simple board of 4"  $\times$  4" in area containing IMB bubble memory module as well as all semiconductor components has already appeared [25]. Research work on 1  $\mu$ m and even 0.5  $\mu$ m bubble diameter materials (potentially up to  $10^6$  bits/in<sup>2</sup> density) have been reported by IBM Research. The manufacturers must get ready to build devices using such materials. Investigators will continue their search for materials sustaining even smaller bubbles. Alternatively, the engineers may invent and implement device structures capable of higher densities (e.g. bubble lattices) than conventional structures (e.g. half disk types used in TIB 0303) at the same bubble diameter.

Similar advances in design are taking place in LSI semiconductor devices. One example is TI's three-dimensional MOS RAM cell design in 1978 that reduces area, power, and refresh requirements. Also, several new semiconductor materials are being discovered, such as Gallium Arsenide, that reduce area and power requirements.

### 2) High resolution lithography

Bubble chips entered the market using high-resolution photolithography (in fact, close to the limit of its capability). Electron beam lithography will reduce the line width by at least another order of magnitude. When used with small-bubble materials or various semiconductor devices, it will enable bit density increase by two orders of magnitude. Again, clever device structure (e.g. contiguous disks or three dimensional MOS devices) achieves higher device density at a given lithography capability, thus providing an alternative to high-resolution lithography.

### 3) Packaging

Packaging considerations can have a large impact on cost, speed, and reliability. Cost, speed, and reliability have and will continue to be substantially improved by putting more devices on a chip. Improvements in device design, better yields to allow larger chips, and higher resolution lithography are increasing the number of devices on a chip at such a drastic rate that it is difficult to comprehend. However, to exploit this requires equally drastic architectural approaches to insure that the number of LSI is minimized. The simple-minded approach of integrating more of the conventional architectures on a chip usually increases the number of pins

per chip beyond cost-effective technological limits (currently about 40 pins per chip). Two approaches can be taken to improve the situation. One approach is to reduce the cost of more pins per chip. Another approach is to reduce the number of pins per chip using a different architectural approach.

Many improvements have been made or proposed to reduce the cost of more pins per chip. Gang bonding and film carrier techniques allow more of the packaging of chips to be automated with improved reliability. Also, putting multiple chips on a single substrate can reduce the cost of packaging. Another technique called wafer-scale integration (WSI) can potentially avoid much of the packing costs by interconnecting the chips directly on the original wafer. Bad chips are removed using laser trimming or using dynamic diagnostic algorithms to locate and electronically disconnect bad chips. The dynamic approach has the advantage that it can be applied to remove chips that have gone bad in installed equipment.

Alternatively, new architectures can cluster hardware onto chips in ways that reduce the number of pins per chip as well as simplifying the interconnection among chips. The cellular-logic devices described in Section III use a one-dimensional array, a tree, or a network. A one-dimensional array requires the fewest pins per cell because each cell need only communicate to its two adjacent cells. Also, the number of pins per chip is independent of the number of cells per chip. This allows the drastic increase in devices per chip to be directly exploited without increasing the number of pins per chip. For example, if one cell per chip requires 16 pins, then 100 cells per chip would require only 16 pins. This advantage also carries over to larger packages, such as printed circuit boards, multiple chip package, and wafer-scale integration. No other topology has this property. All others must increase the number of pins per chip as more cells are integrated into one chip. In order to exploit this advantage, however, the memory and processor of each cell must be compatible technologies, so that they can be packaged (or preferably processed) together. Various semiconductor memory technologies have very compatible logic technologies. Also, magnetic bubble logic shows great promise for exploiting bubble memories. Disc and tape memories, however, have no compatible logic technologies.

The industry has already paid attention to board compatibility and voltage compatibility of bubble components with semiconductor components. Some remaining problems for bubbles with major improvement potentials are multiple-chip packaging, replacement of external bias magnets by on-chip bias, replacement or simplification of the external driving coils, and further development of bubble logic.

### 4) System innovation

The hardware problems are reasonably well defined and being pursued. The system problems are desperately in need of innovation, discipline, and interaction with hardware know-how. There have been enough scattered conceptual explorations of bubble device capabilities (e.g., a variety of device structures for Boolean logic, text editing, data man-

agement, sorting, associative search, etc.). Evaluation of the feasibility of these devices is lacking. No serious commercial impact is foreseen without the development of a few (indeed very few) basic chip types encompassing a collection of universal functions. System assessments are equally lacking. Detailed designs to include system performance evaluation and software requirements are needed to demonstrate the advantages of the innovative hardware designs. As usual, a multi-disciplinary area tends to become a no-man's land. Only simple problems such as simulation and performance evaluation of bubbles and CCD's as gap fillers have been examined, probably over-worked.

Tomorrow's DBM's will depend heavily on both loosely and tightly coupled inter-processor architectures. Communication considerations will begin to dominate price and performance. Realization of DBM architecture will depend heavily on progress in this area.

The design of special purpose LSI devices to fit DBM idiosyncrasies will depend heavily on cutting design and engineering costs for such devices. If costs continue to run high, the DBM implementors will have to structure their thinking toward utilization of more conventionally organized memory and microprocessor components.

### 5) Technology and standardization

Standardization usually comes after developments in products have been done, not before. However in the age of very large scale integration (VLSI), when design cost overshadows manufacturing cost (e.g., see Moore [39]), it would make great sense for the users to indicate what they want to see in the hardware. By adjusting their requirements to the manufacturing constraints of hardwares, they may forecast the standards before the product development, both for user convenience and for manufacturing cost reduction.

Let us clarify the issues by considering a specific technology—magnetic bubbles. At present, bubble memory modules with capacity ranging from 92kb to 1Mb are available commercially. Certainly, the technology is mature enough to consider standardization issues. In the U.S.A., bubble products are marketed by Texas Instruments, INTEL, Rockwell International, and National Semiconductors, and also produced by Western Electric and other companies for internal use. In Japan, Fujitsu, Hitachi, and NEC are manufacturing bubble modules as commercial products (see Yamagishi [53]). Certainly, there are enough manufacturers to make standardization issues relevant and urgent from the user's viewpoint. Moreover, steady improvements of device density and chip capacity have been predicted, and various functional enhancements have been proposed. Certainly, the technology will undergo highly dynamic evolutionary stages and need standardization to prevent unbridled developments.

The maturity of manufacturing technology will encourage the pursuit of associative search, sorting, data management, simple Boolean logic, etc. (see Chang [63]). Although the detailed device configurations must await the gradual hardware evolution, the terminal characteristics of the chips of concern to the users could be responsive to the users, and

early interactions between the manufacturers (or their fore-runners—the researchers and developers) and the users will be worthwhile. Some proposals for standardization may be a reasonable way to initiate the dialogues.

### Hardware architecture

1) Clearly, the proper mix of families of device architectures and speeds will be a major concern of DBM technologists in the '80's. Because of the expense of prototyping such systems, there will be a heavy reliance on modeling and performance evaluation simulations.

2) The need to define logical interfaces and protocols for I/O architectures will become a dominant theme in the '80's [38]. This will be required so that the systems can more easily incorporate various DBM components into integrated systems to meet user application needs. One can anticipate the same controversies to arise in this area as have occurred in communication and networking standardization efforts.

3) The success of category 1 and category 4 DBMs will depend heavily on being able to optimize their usage in broad application environments. For example, they appear to be most cost effective where searching requires complex relationships be satisfied on secondary keys and when multiple records respond to such requests. This feature is expected to become more important in the future when applications are hypothesized to rely heavily on on-line queries. Nevertheless, these devices will have greater applicability if they can also efficiently search for single records. The ability to handle many data types of varying lengths would also broaden their market.

4) The protection mechanism required by databases to control concurrency, security, integrity, and recovery have barely been considered by workers in DBM technology. This is often passed off as a software problem. A fruitful area for DBM researchers will be in designing DBM architecture to support these functions. The inherent speed of associative processors indicates that enforcement of protection rules may become one of their primary functions.

### Software architecture

1) Because database machines will incorporate many diverse processors, bulk memories, and intelligent memories with varying price, performance, and capacity, an extensive amount of work will continue to be needed in studying data clustering, partitioning, staging, and virtual memory strategies for files. Magnetic disks are not likely to disappear in the '80's. Also, other low price/bit large file technologies may come of age in the '80's, e.g., laser video disks and EBAM. They will be used to store the majority of on-line data. Accessing strategies will continue to optimize resources by attempting to minimize the number of disk accesses required to complete an operation. Algorithms that use intelligent controllers and associative memories will be sought to improve access for these bulk memories.

2) An important contribution that is needed to unify database machine research will be the identification of com-

monality and compromise between the individual requirements of text, formatted files, signal, graphic, and map databases.

3) An important issue raised in the past is whether or not database machines should be user programmable. That is, should software be provided to allow users to code data processing and systems programs or should the system limit itself to the execution of database management functions. Precluding the ability to run machine or compiled code will eliminate many of the mechanisms or avenues that allow database security and integrity breaches today. It will also increase the designer's degree of freedom in customizing the DBM for its intended function.

4) The collection and dissemination of user statistics relating to query complexity, file characteristics, locality of database access, etc., are currently non-existent. Without this data, researchers can only hypothesize the relative importance of various architectural tradeoffs. We cannot deliver good solutions until the problems are well understood and parameterized. On the other hand, we cannot parameterize user statistics until we deliver good solutions. Users adapt to whatever system is available. Any statistics gathered from existing systems is only valid pastbound and may not have any resemblance to the future. Improvement must be made iteratively. Because of improvements in hardware, new and improved system strategies will be developed and used. This will, in turn, provide feedback to aid in further hardware improvements.

## VI. CONCLUSION

What impact do hardware technologies and database machines have on the database management area? The answer is: They are all making data processing less expensive and more accessible (to both large and small users). The low-cost, computational, logic and control capabilities have already made microprocessors ubiquitous. Bubbles and CCD's offer modular storage coupled with data storing, arranging and managing capabilities. Their impact will be twofold: First, they will extend database management capabilities to smaller data collections for smaller users in smaller machines. Second, they will be useful in large database systems as nodes in a network, as servers, and as components amenable to parallel operations.

Advances in database machine technology will be required to solve many database management system problems so that the promise of the database gospel can be delivered to users. Progress toward producing these machines will depend heavily on the improvements in price/performance of basic memory and processor technologies. A better understanding of the partitioning of the total problem will also aid special device development. The trend will be toward defining integrated database machines. Thus, workers in this area will find it necessary to have a good understanding of database application and software issues, as well as hardware architecture and technology issues.

The advances in DBM technology will not only have great impact on the implementation of DBMS software but also have profound effect on the designs of DBMS architectures,

data models, and data languages. Database machines can make it very cost-effective to support high-level data models and data languages which are necessary for improving user/programmer productivity and to support multi-schema DBMS architectures which are necessary for achieving data independence. The existing database machines have demonstrated their capabilities to make data mapping between schemas a simpler task and to support the existing data models with considerable improvement in cost/performance. Furthermore, database machines are particularly suitable for supporting high level, non-procedural, and set oriented data languages. Thus, we should establish a standard DBMS architecture or a data model based on user benefits and assume with confidence that the performance gap will gradually close up. High level, non-procedural and set oriented operations which score high in both user productivity and technology considerations should be incorporated in a standard data language.

## REFERENCES

1. Banerjee, J., Hsiao, D. K., and Kannan, K., "DBC—A Database Computer for Very Large Databases," *IEEE Transactions on Computers*, Vol. C-28, No. 6, June 1979.
2. Batcher, K. E., "STARAN Series E," *Proc. 1977 International Conference on Parallel Processing*, Aug. 1977, pp. 140-143.
3. Baum, R. I., Hsiao, D. K., and Kanan, K., "The Architecture of Database—Part I: Concepts and Capabilities," The Ohio State University Technical Report No. OSU-CISRC-TR-76-1, (September, 1976).
4. Berra, P. B. and Oliver, E., "The Role of Associative Array Processors in Data Base Machine Architecture," *Computer*, Vol. 12, No. 3, March 1979.
5. Canady, R. H., Harrison, R. D., Ivie, E. L., Ryder, J. L., and Wehr, L. A., "A Back-End Computer for Database Management," *Communications of the ACM*, 17, 10, (October 1974), pp. 575-582.
6. Chang, H., *Magnetic Bubble Memory Technology*, Marcel Dekker, 1978.
7. Chang, H., "On Bubble Memories and Relational Data Base," *Proc. 4th Int'l Conf. on Very Large Data Bases*, Berlin, Sept. 13-15, 1978, pp. 207-229.
8. Codd, E. F. and Date, C. J., "Interactive Support for Non-Programmers: The Relational and Network Approaches," IBM Research publication RJ1400, San Jose, June 1974.
9. *Computer*, Vol. 12, No. 3, March 1979.
10. Copeland, G. P., "String Storage and Searching for Data Base Applications: Implementation on the INDY Backend Kernel," *Proc. Fourth Workshop on Computer Architecture for Non-Numeric Processing*, SIGARCH SIGIR SIGMOD, Aug. 1978, pp. 8-17.
11. Copeland, G. P., Lipovski, G. J., and Su, S. Y. W., "The Architecture of CASSM: A Cellular System for Non-numeric Processing," *Proc. 1st Annual Symposium on Computer Architecture*, Dec. 1973, pp. 121-128.
12. Davis, E. W., "STARAN Parallel Processor System Software," *AFIPS Conf. Proc.*, Vol. 43, 1974 NCC, pp. 16-22.
13. DeFiore, C. and Berra, P. B., "A Data Management System Utilizing an Associative Memory," *AFIPS Conf. Proc.* Vol. 42, 1973 NCC, pp. 181-185.
14. DeFiore, C. R. and Berra, P. B., "A Quantitative Analysis of the Utilization of Associative Memories in Data Management," *IEEE Trans. Computers*, Vol. C-23, No. 2, 1974, pp. 121-132.
15. DeWitt, D. J., "DIRECT—A Multiprocessor Organization for Supporting Relational Data Base Management Systems," *IEEE Transactions on Computers*, Vol. C-28, No. 6, June, 1979, pp. 395-406.
16. Fisher, P. S. and Maryanski, F. J., "Design Considerations" in *Distributed Data Base Management Systems*, TR CS 77-08, Dept. of Computer Science, Kansas State University, Manhattan, Kansas 66506, April 1977.
17. Freen, R., "A Partitioned Data Base For Use With a Rational Associative Processor," M. S. Thesis, Department of Computer Science, University of Toronto, December 1977.

18. Hakozaki, K., et al., "A Conceptual Design of a Generalized Database Subsystem," *Proc. of the 3rd Int'l. Conf. on Very Large Data Bases*, Oct. 1977, pp. 246-253.
19. Housh, R. D., "A User Transparent Distributed DBMS," Masters Report, Dept. of Computer Science, Kansas State University, Manhattan, Kansas 66506.
20. Hsiao, D. K. and Kanna, K., "The Architecture Of A Database Computer—Part II: The Design of Structure Memory And Its Related Processors," The Ohio State University, Tech Rep. OSU-CISR-TR-76-3 (December 1976).
21. Hsiao, D. K. and Kannan, K., "The Architecture Of A Database Computer—Part III: The Design Of The Mass Memory And Its related Components," The Ohio State University, Tech. Rep. OSU-CISR-TR-76-3 (December 1976).
22. Hsiao, D. K., Kannan, K., and Kerr, D. S., "Structure Memory Designs For A Database Computer," *Proceedings of ACM 77* (October 1977).
23. Hsiao, D. K., Kanna, K., and Kerr, D. S., "Structure Memory Designs for a Database Computer," *Proc. ACM 1977*, Dec. 1977, pp. 343-350.
24. *IEEE Transactions on Computers*, Vol. C-28, No. 6, June, 1979.
25. INTEL Corp., "INTEL Magnetics Bubble Memory Design Handbook," May 1979.
26. Jeffery, S. and Berg, J. L., "Developing a Strategy for Federal DBMS Standards," *Tenth Annual Conf.*, Society for Management Information Systems, Washington, D. C., Sept. 18-20, 1978.
27. Jeffery, S., Fife, D., Deutsch, D., and Sockut, G., "Architectural Considerations for Federal Database Standards," *Spring COMPCON 79*, San Francisco, Calif., Feb. 26-March 1, 1979.
28. Kannan, K., Hsiao, D. K., and Kerr, D. S., "A Microprogrammed Key-work Transformation Unit For A Database Computer," *Proceedings of MICRO-10 Conference*, October 1977.
29. Kuck, D. J., "ILLIAC IV Software and Application Programming," *IEEE Transactions on Computers*, Vol. C-17, No. 8, August 1960.
30. Lin, C. S., Smith, D. C. P., and Smith, J. M., "The Design of a Rotating Associative Memory for Relational Data Base Applications," *ACM Trans. Database Systems*, Vol. 1, No. 1, 1976, pp. 53-65.
31. Linde, R., Gates, R., and Peng, T. F., "Associative Processor Applications to Real-time Data Management," *AFIPS Conference Proceedings*, Vol. 42, 1973, pp. 187-195.
32. Lipovski, G. J., "Architectural Features of CASSM: A Context Addressed Segment Sequential Memory," *Proc. 5th Annual Symposium on Computer Architecture*, Palo Alto, Calif., April 1978, pp. 31-38.
33. Lowenthal, E. I., "The Backend Computer, Part I and Part II," Auerbach (Data Management) Series, 24-01-04 and 24-01-05 1976.
34. Lowenthal, E. I., "A Survey: The Application of Data Base Management Computers in Distributed Systems," *Proceedings of the Third International Conference on Very Large Data Bases*, Tokyo, October 1977.
35. Madnick, S. E., "INFOPLEX—Hierarchical Decomposition of a Large Information Management System Using a Microprocessor Complex," *Proc. 1975 NCC*, Vol. 44, AFIPS Press, Montvale, N. J., pp. 581-586.
36. Marill, T. and Stern, D., "The Data Computer—A Network Data Utility," *1975 NCC*, Vol. 44, June 1975.
37. Maryanski, F. J. and Wallentine, V. E., "A Simulation Model of a Backend Data Base Management System," *Proceedings 7th Pittsburgh Symposium on Modeling and Simulation*, pp. 252-257, April 1976.
38. McDonnell, K., "Trends in Non-Software Support For Input-Output Functions," *Proc. of the 3rd Workshop On Computer Architecture for Non-Numeric Processing*, May 1977 40-47.
39. Moore, G., "VLSI: Some Fundamental Challenges," *Spectrum*, Vol. 16, no. 4, April 1979.
40. Moulder, R., "An Implementation of a Data Management System on an Associative Processor," *AFIPS Conf. Proc.* Vol. 42, 1973 NCC, pp. 171-176.
41. Ozkarahan, E. A., Schuster, S. A., and Smith, K. C., "RAP—An Associative Processor for Data Base Management," *AFIPS Conf. Proc. 1975 NCC*, pp. 370-387.
42. Rosenthal, R. S., "An Evaluation of a Backend Data Base Management Machine," *Proceedings of the Annual Computer Related Information Systems Symposium*, U. S. Air Force Academy, 1977.
43. Rudolph, J. A., "A Production Implementation of an Associative Processor: STARAN," *AFIPS. Conf. Proc. 1972 FJCC*, Vol. 41, Part I, pp. 229-241.
44. Schuster, S. A., Ozkarahan, E. A., and Smith, K. C., "A Virtual Memory System for a Relational Associative Processor," *Proc. Nat. Computer Conf.*, 1976, pp. 855-862.
45. Schuster, S. A., Nguyen, H. B., Ozkarahan, E. A., and Smith, K. C., "RAP.2—An Associative Processor for Databases and Its Applications," *IEEE Transactions on Computers*, Vol. C-28, No. 6, June 1979, pp. 446-458.
46. Slotnick, D. L., "Logic per Track Devices," in *Advances in Computers*, Academic Press, 1970, pp. 291-296.
47. Su, S. Y. W., "Cellular-logic Devices: Concept and Applications," *Computer*, Vol. 12, No. 3, March 1979, pp. 11-25.
48. Su, S. Y. W., Copeland, G. P., and Lipovski, G. J., "Retrieval Operations and Data Representations in a Context-addressed Disc System," in *Proceedings of ACM's SIGPLAN and SIGIR Interface Meeting*, Nov. 1973, pp. 144-156.
49. Su, S. Y. W., Nguyen, L. H., Emam, A., and Lipovski, G. J., "The Architectural Features and Implementation Techniques of the Multicell CASSM," *IEEE Transactions on Computers*, Vol. C-28, No. 6, June, 1979, pp. 430-445.
50. Su, S. Y. W., "Associative Programming in CASSM and its Applications," *Proc. of the Third International Conference on Very Large Databases*, Oct. 6-8, 1977, pp. 213-228.
51. Su, S. Y. W., Lupkiewicz, S., Lee, C. J., Lo, D. H., and Doty, K., "MICRONET: A Microcomputer Network System for Managing Distributed Relational Databases," *Proc. of the 4th International Conference on Very Large Data Bases*, Berlin, Germany, Sept. 13-15, 1978.
52. Tschritzis, D. and Lochovsky, F., *Data Base Management Systems*, Academic Press, 1977.
53. Yamagishi, K., "The Progress of Magnetic Bubble Development in Japan," *Proc. 3rd U.S.A.-Japan Computer Conference*, October, 1978.

# CONLAN—A formal construction method for hardware description languages: basic principles

by ROBERT PILOTY

*Technische Hochschule Darmstadt, FR Germany*

MARIO BARBACCI

*Carnegie-Mellon University*

DOMINIQUE BORRIONE

*Universite de Grenoble, France*

DONALD DIETMEYER

*University of Wisconsin-Madison*

FREDRICK HILL

*University of Arizona-Tucson*

and

PATRICK SKELLY

*Honeywell, Phoenix*

## 1. INTRODUCTION

The development of a CONLAN(CONsensus LANguage) goes back to the first Symposium on Hardware Description Languages (HDL) at Rutgers University in 1973. It was initiated by J. Lipovski, then Univ. of Florida. After two years of preparatory work the CONLAN Working Group was formed on the occasion of the third Symposium on HDL in New York. These papers represent the result of four years of hard work of a group spread out over two continents. This work is by no means complete; many things have still to be done. Nevertheless, encouraged by the positive response to an informal presentation of our approach at the fourth Symposium on HDL in Palo Alto 1979, we feel that publication of what we have obtained so far is warranted. This paper presents the basic principles of CONLAN. Two companion papers [1,2] treat language derivation and language application within the framework of CONLAN. A more detailed report, of which a draft exists already, will be forthcoming soon.

## 2. MOTIVATION AND OBJECTIVES

The decision to start the CONLAN project was motivated by the following assessment of the situation in the area of HDL and of Computer Aided Design (CAD) tools based on them:

Several dozen HDL's existed in 1973 [3] and every year

since new languages have been proposed and published, mostly from persons in academic institutions [4,5,6]. This tendency to proliferation is in sharp contrast to acceptance in industry. Neither have they been used to document the design process of digital systems nor to support tools for certification, synthesis, and performance evaluation to any appreciable extent. Most CAD tools in industry are designed to aid the manufacturing process (placement, routing, mask layout etc.). The process of systems and logic design is mostly carried out in the traditional way of drawing block and circuit diagrams at the IC package or gate level. In many cases these diagrams are the only true and complete documentation of the system. Most other aspects or phases of the system design, particularly system behavior, are informally and incompletely described. Simulation as a means for advanced certification is used, if at all, at a very low level (mostly gate level) and hence at enormous cost for more complex systems. Most of the certification is done very late at the level of a physical prototype causing costly changes in physical design.

This situation has not changed very much in the four years of CONLAN development: HDL's continue to proliferate [7,8] but their usage in real life design has not increased in the same proportion. Only recently a growing interest in efficient tools for design support at systems and logic level can be observed, probably due to the advance of LSI and VLSI, where late changes make a system more and more costly, and due to increased system complexity in a competitive market, calling for more efficient design tools [9,10].

There are several reasons why acceptance of existing HDL's is so low:

1. None of the languages alone is of sufficient scope to portray all aspects of a system and cover all phases of the design process.
2. Languages of different scope are syntactically and semantically unrelated.
3. Few of the languages are formally defined.
4. Only a few languages are implemented.
5. Descriptions are represented by character strings rather than diagrams.
6. There exists no comprehensive hardware and firmware design methodology telling how to use HDL's effectively.

The main aim of the CONLAN Working Group is to remedy the first four deficiencies [11]. Its primary objectives are:

1. to provide a common formal syntactic and semantic base for all levels and aspects of hardware and firmware description, in particular for descriptions of system structure and behavior.
2. to provide a means for the derivation of user languages from this common base
  - having a limited scope adjusted to a particular class of design tasks,
  - thus being easy to learn and simple to handle,
  - yet having a well defined semantic relation among each other.
3. to support CAD tools for documentation, certification, design space exploration, synthesis and so on.

CONLAN is not intended as a language standard trying to impose a certain style of hardware description on makers of design tools. It should rather be viewed as a formal system which allows them to construct HDL's of their choice in a consistent and unambiguous way within some notational conventions.

Character strings using the ISO-IRV 646 character set are the basic means of representing CONLAN descriptions, since they are more general and easier to use as an input or output for CAD tools. However it is an objective of the CONLAN Group at a later stage, to show how to write descriptions in CONLAN which are isomorphic to space structure (network descriptions) or behavior (sequential and concurrent flow descriptions), and to propose graphical members of the CONLAN family.

Problems of design methodology are not treated in this presentation of CONLAN. Much remains to be done in this area although a number of significant results have been obtained already.

### 3. BASIC APPROACH

#### *Language family*

CONLAN supports a *self-defining, extensible* family of languages. Its member languages are tied together by a com-

mon core syntax and a common semantic definition system. The CONLAN construct to define a member language is called a *language definition segment*.

#### *Descriptions*

The member languages are used to write descriptions of hardware, firmware or software modules. *Description definition segments* are provided for this purpose as a CONLAN construct.

#### *Abstract datatypes*

The method for semantic definition is based on the concept of abstract data types, which has been developed for programming languages like CLU [12,13] or ALPHARD [14,15]. An abstract data type, henceforth called a *type*, is defined by a domain of elements and a set of operations on these elements. New types may be defined in terms of primitive types supplied with the language.

#### *Reference language*

Contrasting the application of types in programming languages, the primitive types of a CONLAN member language except Primitive Set CONLAN (described later in this paper) are not implied. Rather, they are defined in terms of the types of one other member language. This language is called the *reference language* of the language being defined. This establishes a partial order among the member languages: A language  $L_b$  is derived from a language  $L_a$  if  $L_a$  is the reference language of  $L_b$  or if there is a chain of reference languages leading from  $L_b$  to  $L_a$ .

#### *Self-definition*

In CONLAN the same construction mechanism and the same notational system used to provide descriptions is also used to define new language members. In this sense CONLAN is self-defining in contrast to externally defined languages using a separate language to define its semantics, e.g., the formal description of PL/I using the Vienna Definition Language [16].

#### *Extensibility*

The CONLAN family of languages is open ended. New languages may be derived from existing ones at any given point in time as the need arises. They in turn may be used later as reference languages for further languages.

#### *Syntax modification—core syntax*

The syntax of a new CONLAN member may be made to differ from the syntax of its reference language by adding

and/or deleting productions in the reference syntax. A FORMAT statement is provided for this purpose. This capability permits the language designer to keep the syntax of a new language as simple as possible and yet allows the incorporation of new constructs to denote specific features, e.g., the introduction of an infix symbol to denote some new operation. There is a set of productions which may never be deleted, and thus is common to all CONLAN members. It is called the *core syntax*.

#### CONLAN text structure

The CONLAN text structure is shown in Figure 1. At any given point in time it consists of a set of language definition segments and a set of description segments. Each segment is under the scope of a REFLAN statement. In a language definition segment this statement points to the language from which the new language is directly derived. Thus language LL1 is directly derived from L1, and LL2 is directly derived from L2. In a description segment the REFLAN statement points to the language in which the description is written.

#### Hiding of types and operations

It is important to note the CONLAN concept of hiding types and operations.

Referring to Figure 1, for the writer of a LL1 only those types and operations which are defined in L1 and not marked PRIVATE are visible and accessible. This implies that the types and operations of bcl are inaccessible to LL1 unless explicitly brought forward by L1 with a CARRY statement. The CARRY statement avoids the need for redefinition if a type is used in more than one language level. This hiding mechanism is the main instrument to keep derived languages simple and maintain a clear semantic relation with their ancestor languages.

#### Base conlan—toolmakers, users

There is one root language called base conlan (bcl), serving as the interface between the CONLAN Working Group and its public of *toolmakers* and *users*. Toolmakers start from bcl to construct languages and their associated CAD tools. Users write descriptions of hardware and firmware systems in these languages. Bcl provides a carefully chosen set of basic types reflecting the CONLAN concept of time and space, of signals and carriers, of arrays and records. It is described in more detail in Reference 1.

#### Primitive set conlan

To define bcl the CONLAN Working Group used a very low level but powerful language called primitive set conlan (pscl) to formally define the concepts represented by the bcl types. Pscl has no reference language. It owns a set of primitive types whose domains and operations are introduced

informally. The rest of this paper is devoted to the formal concepts of CONLAN and a description of pscl.

## 4. FORMAL CONCEPTS

CONLAN deals with the following categories of objects

—Elements, Parameters, Operations, Types and Classes, Descriptions, and Languages

Identifiers are used to denote CONLAN objects. *Simple identifiers* start with at least one letter followed by an arbitrary string of letters, digits, and underscore ('\_'), of any length and terminating with a letter, or digit, or with the symbol @. Symbol @ denotes a system identifier which may be used only within a language definition segment. *Compound identifiers* are two or more simple identifiers separated by period ('.'). They permit one to prefix a simple identifier with one or more enclosing segment identifiers if the corresponding object is referenced outside the segment providing the simple identifier.

### 4.1 Elements

Elements are the operands for CONLAN operations. CONLAN works with a well defined universe of elements. Subsets of this universe serve as domains for the operations.

### 4.2 Parameters

There are formal and actual parameters. A formal parameter is denoted by an identifier representing any element of a given type. Formal parameters appearing in a parameter list of an operation or type or a description are typed, i.e., the parameter is followed by the type designator separated by a colon (e.g., x:bool). The designator defines the domain of x and the operations applicable to it.

Typing of formal parameters permits type checking: When a formal parameter is bound to an actual parameter, its associated type designator is used to check if the type of the actual parameter is equivalent to the type of the formal parameter. The type of the actual parameter is normally explicitly stated in a declaration, or in one of the constructs involving predicates (e.g., parameter a in ALL a:int WITH pred(a,u,v) ENDALL). On the other hand, when the actual parameter is a constant denotation, it can also be checked to determine that it belongs to the domain of the prescribed type.

Generic segments [17] may be specified in CONLAN, i.e., segments which accept an operation identifier (operator symbol) or a type designator as a parameter. Formal generic parameters must be typed using keyword FUNCTION or ACTIVITY or a class designator. Consider for example:

$x(\dots, f:\text{FUNCTION}(\text{int}, \text{int}):\text{bool}, \dots)$  or  $x(\dots, u:\text{someclass}, \dots)$

In the first case *f* is typed to accept any function identifier which is defined as a binary function with an integer domain



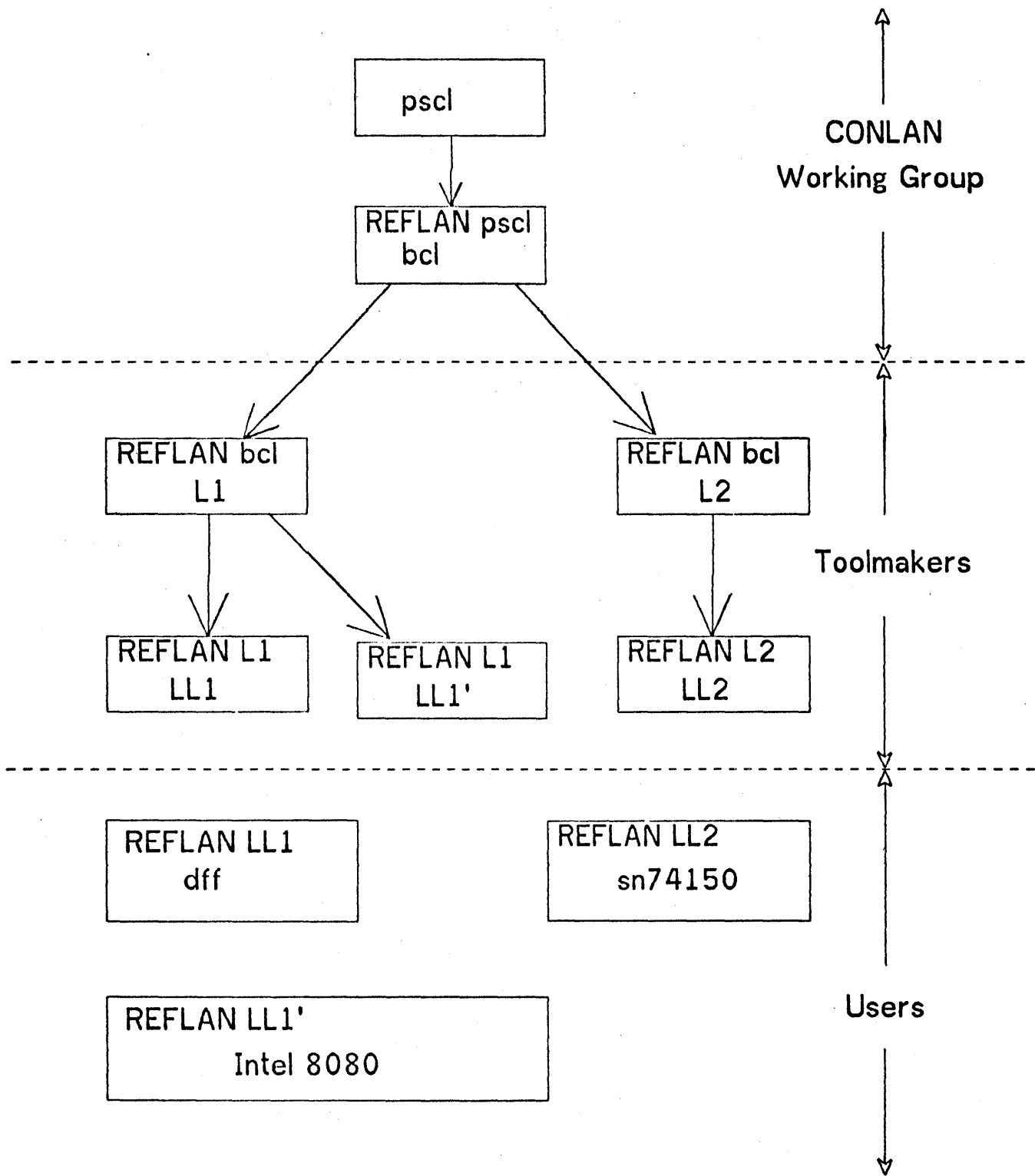


Figure 1—CONLAN text structure.

and a Boolean range. In the second case  $u$  accepts any type designator, which belongs to the class 'someclass.'

### 4.3 Operations

CONLAN operations are normally denoted by an identifier prefixing a list of parameters. Operator symbols may be introduced for prefix and infix notation either in lieu of a standard functional notation or as an alternative to it via syntax modification.

Two categories of operations are known in CONLAN: *activities* and *functions*.

#### Activities

An activity changes the state of one or more carriers. It is defined by an *activity definition segment* of the form:

ACTIVITY ident (typed-parameters) BODY.....ENDident

Formal parameters are typed. The body may contain declarations for local carriers and enclose a non-empty list of invocations of previously defined activities.

An activity may be invoked in five modes:

- unconditional, e.g.: act1(exp1,exp2,...)
- IF c1 THEN act1 ELIF c2 THEN act2....ELSE actn  
ENDIF
- ON c1 IS act1 ENDON
- CASE exp IS x1:act1,...x<sub>n</sub>:actn, ELSE act  
ENDCASE
- ONCASE exp IS x1:act1,...x<sub>n</sub>:actn, ELSE act  
ENDON

If invoked under IF or CASE it is evaluated as long as the condition is true. If invoked under ON or ONCASE it is evaluated whenever a change in the value of the condition is detected.

#### Functions

A function maps a domain of elements into a range of elements. It is defined by a *function definition segment* of the form:

FUNCTION ident(typed-parameters): result-type  
BODY....RETURN expr ENDident

Formal parameters are typed. The body may contain declarations for local carriers and enclose a non-empty list of invocations of previously defined activities.

Functions are normally invoked in expressions. Expressions returning a Boolean result may also be called predicates. A new expression may be formed from existing ones in one of the following forms.

- nesting, e.g. f1(exp1,exp2,...)
- IF c1 THEN exp1 ELIF c2 THEN exp2.... ELSE  
expn ENDIF
- THE@ r:restype WITH@ pred(r,a,b...) ENDTHE

The last form selects the element  $r$  from the result type for which the predicate becomes true. If it becomes true for none or several elements of *restype* an error condition exists.

To construct new predicates from existing predicates the equivalent of quantifiers known from first order predicate calculus is available in CONLAN. The following forms may be used in addition to those of the preceding paragraph:

- FORALL@ x:atype IS@ pred(x,a,b...) ENDFOR
- FORSOME@ x:atype IS@ pred(x,a,b...) ENDFOR
- FORONE@ x:atype IS@ pred(x,a,b...) ENDFOR

Assertions are predicates which denote conditions on actual input, output and attribute parameters as well as on local objects. Assertions must be true at every point in time when the segment containing these assertions is invoked (operations) or used (descriptions). If an assertion is not true an error condition exists.

### 4.4 Types and classes

Types are defined in *type definition segments* of the following form (simplified):

TYPE t2(typed-parameters) BODY set-definition carry  
operation-definition ENDT2

Parameters are optional. If present, a *family of types* is specified by the definition.

The set definition part serves to specify the domain of the type. It may be specified in any of three ways:

1. as a true subset of the domain of an existing type  $t_1$  using the subset constructor:

ALL x:t1 WITH pred(x,a,b) ENDALL

2. identical to the domain of an existing type by simply writing its designator,  $t_1$ ;
3. by enumeration of constant denotations.

In cases (1) and (2) type  $t_1$  is called the *defining type*. In case (3) the defining type is the universal type  $univ@$ .

In the operation definition part, operations may be defined to operate on the new type. These operations are defined in terms of the operations previously defined including operations of the defining type  $t_1$ . The operations of the defining type  $t_1$  may not be applied to the domain of the defined type  $t_2$  outside the body of the definition of  $t_2$  (hiding of operations) unless explicitly listed in the carry part. The parameters or results of the carried operations originally typed with  $t_1$  are then considered typed with  $t_2$  (implicit type conversion).

Type designators, may be used as operands in type relations:

1. Two types  $s$ ,  $t$  are *equal* ( $s=t$ ) if  $s$  and  $t$  refer to the same type definition segment.
2. Let  $s$  and  $t$  designate two types. Then  $t$  is *derived from*  $s$  ( $t<|s$ ) iff  $s=t$  or  $t$  is a subtype of  $s$  or  $s$  is the defining type of  $t$  or there exists a set of types  $t_m, t_{m-1}, \dots, t_i, \dots, t_1$ , for  $m>2$  such that  $t_m=t$  and  $t_1=s$  and  $t_i<|t_{i-1}$  for all  $i=2, \dots, m$ .
3. Let  $x$  represent an element of the CONLAN universe and  $t$  the designator of any type in CONLAN. Then  $x$  is an *element of*  $t$  ( $x \in t$ ) if  $x$  is the denotation of an element of the domain of  $t$ .

Classes are named sets of type designators together with operations defined on elements of that set. A class is defined by a *class definition segment* analogous to a type definition. The class whose domain consists of the type designators of all types defined or yet to be defined is called the universal class **any@**. The main use of class definitions is to introduce designators for generic parameters, e.g., TYPE array (u:any@).

#### 4.5 Descriptions

A *description definition segment* is used to define the input/output relation of hardware, firmware, or software modules. Details about their definition and usage can be found in Reference 2.

```
DESCRIPTION nand(IN x,y: btml OUT z: btml) BODY
z. = (x-^y)Δ1 ENDnand
```

In the above example,  $\Delta$  is the decay operator,  $.$  = denotes terminal connection, and btml has been defined as the type "Boolean terminal with default value 1" in Reference 1.

#### 4.6 Languages

A CONLAN member language is defined via a *language definition segment* of the form:

```
REFLAN oldlanguage CONLAN newlanguage
BODY...ENDnewlanguage
```

The constituents of the body are:

- list of carried types and operations from the reference language (optional)
- definition of public and private types
- operation definitions (optional)
- description definitions (optional)
- format statements (optional)

Notice that operations may be defined outside type segments. A library of operations could therefore be defined as part of a CONLAN segment. The same applies to descrip-

tions if the language is geared to describing systems in terms of standard modules (e.g. TTL-modules). Syntax modifications via format statements will be illustrated in Reference 1.

All types referenced in the formal parameter lists appearing in non-private type, operation, and description definitions must either be explicitly defined or carried from the reference language (closure of a language with respect to types).

#### 5. PRIMITIVE SET CONLAN (pscl)

Pscl is the lowest language level in CONLAN. It has no reference language. Hence its types (domains and operations) are defined informally.

##### UNIV@

Type **univ@** consists of all members of all types defined or yet to be defined in all members of the CONLAN family, together with operators '=' and '≠'. It permits the present definition of operations on objects to be defined in the future:

```
univ@ = { .., -1, 0, +1, .., '!', .., 0, (.0, 0.), .., 'xYz', .. }
```

Type **univ@** is considered as the defining type for the other types of pscl, namely "int," "bool," "string," "cell@," "tuple@" from which all other types of CONLAN will be derived.

##### ANY@

Class **any@** is the universal class in CONLAN, and the only class known in pscl. Its domain is the set of designators for all types defined or yet to be defined in all members of the CONLAN family, together with operations '=', '≠', 'ε', and '<|'.

```
any@ = {univ@, int, bool, tuple@, cell@, string, ...}
```

Function 'ε' may be used to determine if an object from **univ@** is a member of a defined type (a member of **any@**).

Function '<|' may be used to determine if a member of **any@** (a type) was derived from another member of **any@**.

##### INT

Type **int** consists of all integers, together with a substantial number of operators provided without formal definition, i.e., they are "known." An integer is denoted with a contiguous sequence of symbols, digits and capitals that may be partitioned into the sign part, magnitude part and base indicator. The sign part consists of symbol + (optional) or symbol -. The magnitude may be expressed in decimal, binary (B), octal (O), or hexadecimal (H). For instance,

```
-12 = -1100B = -140 = -CH
```

**BOOL**

Type **bool** has two members, denoted by 1 and 0 representing “true” and “false” respectively, together with operations '=', '≠', '∧', '∨', '¬', '<', '≤', '>', '≥'. The relational operators are based upon 0 being less than 1.

**STRING**

Type **string** consists of all sequences of characters, together with operations '=', '≠', '<', '≤', '>', '≥', and **order@**. The objects of string are denoted by enclosing the sequence in single quotes ('). Sequences such as '1A', 'b+5', are included. The character ' must be doubled if it is to appear in a string denotation (e.g., 'What's his name?').

The relational operators are based in the order of characters in the ISO-IRV 646 standard. When comparing strings of different length, the shorter string is padded or extended with trailing spaces (code 20H).

Function **order@** takes a string as a parameter and returns an integer computed by treating the elements of the string (i.e., the characters) as 'digits' in a base 128 representation. The leading character is the most significant 'digit'. For instance, **order@('Xy2')** returns (58H\*128\*128 + 79H\*128 + 32H), that is, 163CB2H.

**TUPLE@**

Type **tuple@** consists of all lists of members of **univ@**, together with operations '=', '≠', **size@**, **select@**, **remove@**, and **extend@**. **Tuple@** includes the empty list. A tuple is denoted via a list of object denotations enclosed in '(.' and '.)', and separated by commas.

Two tuples are equal ('=') if they have the same size and identical members in identical order. Otherwise they are not equal ('≠').

Function **size@(x)** returns the number of members of a tuple. If the tuple is empty, **size@** returns 0. Consecutive integers from 1 to **size@(x)** identify the positions of the members of tuple *x*. Only the positions of this range may be referenced. Attempts to reference positions outside this range result in an error report.

Function **select@(x,i)** returns the member of tuple *x* in position *i* (if integer *i* is in the range 1 through **size@(x)**).

Function **remove@(x,i)** returns the tuple *y* such that *y* holds all components of *x* in the same order except the *i*th one if *i* is in the range from 1 to **size@(x)** else an error condition exists. If **size@(x) = 1** then the empty tuple is returned.

Function **extend@(x,u)** returns the tuple *y* of size **size@(x)+1** with the leftmost **size@(x)** components being identical to those of *x* and *u* as the component in position **size@(x)+1**.

Tuples are never modified. **Remove@** and **extend@** simply select the member of the set of all tuples with the right size and contents. The original tuple is not altered.

**CELL@**

Type **cell@** consists of all 'containers' of members of **univ@**. Cells can contain at most one element of a type. The type of the contents must be specified as a parameter in a **type\_designator**, when the cell is declared. For instance, **cell@(t:int)**. Cells are initially empty.

Function **cell\_type@(x)** returns a member of **any@**, namely the type of the (potential) contents of cell *x*.

Function **empty@(x)** returns 1 ('true') if cell *x* is empty otherwise it returns 0 ('false').

Function **get@(x)** returns the contents of cell *x*. If the cell *x* is empty an error condition exists.

Activity **put@(x,u)** replaces the contents of cell *x* and *u*. The type of *u* must be identical to the type of the contents of cell *x*. Attempts to put an element of the wrong type result in an error condition. If cell *x* is empty, **put@** simply inserts *u* in the cell.

Cells are potentially modifiable objects (via function **put@**). These are the only objects with this attribute and constitute the bases for the development of carriers, variables, and other modifiable objects in the CONLAN family.

**6. EXAMPLES OF TYPE AND CLASS DERIVATION**

From the types of **pscl** new types have been derived using the techniques explained under type definition. For example,

**6.1 Typed tuples**

**TYPE** **tytuple@(t: any@) BODY**

**ALL** *x*: **tuple@** **WITH** **FORALL** *i*: **bint**(1,**size@**(*x*))

**IS@** *x*[*i*] ∈ *t* **ENDFORALL** **ENDALL**

**CARRY** =, ≠, **size@**, **remove@** **ENDCARRY**

**FUNCTION** **select**(*x*: **tytuple@**(*t*), *i*: **pint**): *t*

**RETURN** **THE@** *z*: *t* **WITH@** *z* =

**select@**(**old@**(*x*),*i*)

**ENDselect**

**FUNCTION** **tail**(*x*: **tytuple@**(*t*)): **tytuple@**(*t*)

**ASSERT** **size@**(*x*) > 1 **ENDASSERT**

**RETURN** **remove@**(*x*,1)

**ENDtail**

**FUNCTION** **extend**(*x*: **tytuple@**(*t*), *a*: *t*) : **tytuple@**(*t*)

**RETURN** **new@**(**extend@**(**old@**(*x*),*a*))

**ENDextend**

“/Swap two elements of a typed tuple/”

**FUNCTION** **exchange**(*x*: **tytuple@**(*t*), *i*, *j*: **pint**):

**tytuple@**(*t*)

**ASSERT** *i* ≤ **size@**(*x*), *j* ≤ **size@**(*x*) **ENDASSERT**

**RETURN** **THE@** *y*: **tytuple@**(*t*) **WITH@**

**size@**(*y*) = **size@**(*x*) ∧

**FORALL@** *k*: **bint**(1,**size@**(*x*)) **IS@**

**IF** *k* = *i* **THEN** *y*[*k*] = *x*[*j*]

```

    ELIF  $k = j$  THEN  $y[k] = x[i]$  ELSE  $y[k] = x[k]$ 
    ENDIF ENDFORALL ENDTHE

```

```

ENDexchange
ENDtytuple

```

Type `tytuple@` is a generic type. Its formal parameter  $t$  may be bound to the designator of any defined type. In its definition two auxiliary types are used which are not explicitly derived here: (1) Type `pint`, whose domain is all positive integers together with all integer operations. and (2) Type `bint` ( $i,j:int$ ), whose domain consists of all sets of consecutive integers  $\{i, \dots, j\}$ .

The domain of type `tytuple@` is the set of all tuples whose components are elements of the same type  $t$ . Functions `=`, `≠`, `size@`, and `remove@` are carried from `tuple@` to `tytuple@`. In addition, three new functions are defined for elements of `tytuple@`: `select`, `extend`, `exchange`.

Function `select(x:tytuple@(t),i:pint):t` on `tytuple@` is defined to be identical to function `select@(x:tuple@,i:int):univ@` on `tuple@` restricted to the types `tytuple@`, `pint` and  $t$ . A special type conversion operator `old@(x)` is used to convert the element  $x$  of `tytuple@` to the old type `tuple@` before the function `select@` of `tuple@` is applied.

Function `extend` on `tytuple@` is equivalent to `extend@` on `tuple@` but restricted in the typing of its parameters and the result.

Function `exchange` forms a tuple  $y$  by interchanging elements  $i$  and  $j$  of tuple  $x$ .

## 6.2 Scalar value types

```

CLASS scalar_value_type BODY
  ALL x:any@ WITH  $x <| int \vee x <| bool \vee x <| string$ 
  ENDALL
  CARRYALL
ENDscalar_value_type

```

Class `scalar_value_type` is defined as the set of those types (i.e., members of `any@`) which are equal to or derived from one of the primitive types `int`, `bool`, or `string`. The `CARRYALL` statement makes all relations defined on `any@` also available to the types belonging to class `scalar_value_type`.

As an example of the use of classes:

```

TYPE scalar_cell(u:scalar_value_type) BODY
  cell@(u)
  CARRYALL
ENDscalar_cell

```

Generic type `scalar_cell` has a parameter  $u$  which may be bound only to a `scalar_value_type`, for example:

```

scalar_cell(bool)

```

specifies cells which may only contain Booleans. Thus writing

```

scalar_cell(tytuple(bool))

```

will result in a type checking error since because `tytuple(bool)` is not an element of the class `scalar_value_type`.

## 7. CONCLUSIONS

In this paper the motivation, objectives, basic concepts of the CONLAN family and its primitive set members are described.

To promote an orderly development of hardware description languages and to enhance their acceptance in an industrial environment, a powerful construction mechanism for such languages, based on a common core syntax is presented. This construction mechanism ensures that the semantics of the languages derived are well defined. Further, semantically related languages can be constructed which permit the description of digital systems at different levels of abstraction. The common core syntax facilitates learning a new language written in the CONLAN framework. In addition, capabilities for syntax modification permit the suppression of unneeded constructs and the introduction of shorthand for frequently used objects to obtain simple yet useful languages.

We feel that the primitive set language, together with the construction mechanism are not only valid for the development of hardware description languages but also represent a contribution to the available techniques for formal semantic specification of operative languages, including programming languages.

The three papers presented in this series illustrate the basic principles of the construction mechanism, the process of language derivation, and examples of the application of a derived language to hardware description.

Additional reports on the progress of the CONLAN working group are in preparation. Future efforts will be directed toward the preparation of a complete report covering our method for syntax modification, the formal development of the CONLAN array and record constructors, and Base CONLAN (the constructional base for user languages), as well as examples of user languages. In addition, the working group expects to develop more comprehensive user languages covering the design of systems at the gate level, register transfer level, instruction set level (macro and micro programming), and system level.

## 8. ACKNOWLEDGMENTS

The authors are indebted to Bell Northern Research (Ottawa), Sperry Univac (Philadelphia), Office of Naval Research, Ballistic Missile Defense Advanced Technical Center (Huntsville), IRIA (Paris), Bundesministerium für Forschung und Technologie (Bonn), Siemens (Munich), and Fujitsu (Tokyo) for their interest and support, Professor Yaohan Chu for his early contributions, and in particular to Professor Jack Lipovski for his help and unwavering confidence in the group.

## 9. REFERENCES

1. Piloty, R., Barbacci, M., Borrione, D., Dietmeyer, D., Hill, F. and Skelly, P., "CONLAN—A Formal Construction Method for Hardware Description Languages: Language Derivation," *Proceedings National Computer Conference*, Volume 49, Anaheim, California, 1980.
2. Piloty, R., Barbacci, M., Borrione, D., Dietmeyer, D., Hill, F. and Skelly, P., "CONLAN—A Formal Construction Method for Hardware Description Languages: Language Application," *Proceedings National Computer Conference*, Volume 49, Anaheim, California, 1980.
3. Barbacci, M. R., "A Comparison of Register Transfer Languages for Describing Computers and Digital Systems," IEEE Computer Society, *Transactions on Computers*, Volume C-24, Number 2, February 1975.
4. Special issue on Hardware Description Languages, IEEE Computer Society, *Computer*, Vol. 7, No. 12, Dec. 1974.
5. *Proceedings of the 2nd International Symposium on Computer Hardware Description Languages*, Darmstadt, ACM German Chapter Lectures W—1974.
6. *Proceedings of the 3rd International Symposium on Computer Hardware Description Languages and their Applications*, New York, Sept. 3-5, 1975. IEEE Cat. No. 75 CH1010-8C.
7. *Proceedings of the 4th International Symposium on Computer Hardware Description Languages*, Palo Alto, Oct. 8-9, 1979 IEEE Cat. No. 79 CH1436-5C.
8. Special issue on Hardware Description Languages, IEEE Computer Society, *Computer*, Vol. 10, No. 6, June 1977.
9. Collection of Proceedings of the IEEE, ACM Design Automation Conference.
10. Collection of Proceedings of the Fault Tolerant Computing Symposia.
11. Piloty, R., "Guidelines for a Computer Hardware Description Consensus Language" (2nd draft), Memorandum to the Conference on Digital Hardware Languages, June 6, 1976.
12. Liskov, B. and Zilles, S., "Programming with Abstract Data Types," SIGPLAN Notices 9, pp. 50-59, April 1974.
13. Liskov, B., Snyder, A., Atkinson, R. and Schaffert, C., "Abstraction Mechanism in CLU," Computation Structures Group, Memo 144-1, MIT January 1977.
14. Wulf, W. A., "Alphard: Toward a Language to Support Structured Programming," Technical Report, Department of Computer Science, Carnegie-Mellon University, April 1974.
15. Wulf, W. A., London, R. L. and Shaw, M., "Abstraction and Verification in ALPHARD," Technical Report, Department of Computer Science, Carnegie-Mellon University, March 1976.
16. Lucas, P. and Walk, K., "On the Formal Description of PL/I," *Annual Review of Automatic Programming*, Vol. 6, part 3, 1969.
17. Jacquet, P., "Les Types Generic: Propositions pour un Mecanisme d'Abstraction dans les Langages de Programmation."



# CONLAN—A formal construction method for hardware description languages: language derivation

by ROBERT PILOTY

*Technische Hochschule Darmstadt*  
FR Germany

MARIO BARBACCI

*Carnegie-Mellon University*  
Pittsburgh, Pennsylvania

DOMINIQUE BORRIONE

*Universite de Grenoble*  
Grenoble, France

DONALD DIETMEYER

*University of Wisconsin-Madison*  
Madison, Wisconsin

FREDRICK HILL

*University of Arizona*  
Tucson, Arizona

and

PATRICK SKELLY

*Honeywell*  
Phoenix, Arizona

## INTRODUCTION

A CONLAN document has significance only if it is read by a person or machine. That reader (environment) is required to use available facilities to respond to and interact with the document. It must provide the type checking mechanism. It must record the names of defined and declared items and provide the data base they require. It must record signal values. From such records, it can determine facts of importance to continued document evaluation. "System interfaces" are prescribed environment responses, not formally defined via CONLAN syntax.

### 1.1 Values, signals, and carriers

Three broad classes of objects are of primary concern in working members of the CONLAN family:

"Values" are static objects; they do not change with time. An integer, a character, etc. are values.

"Signals" are lists of values. A different time is associated with each value. A signal is then a history of values.

"Carriers" are containers for values or signals. These

values or signals can be replaced as a result of an operation invocation.

### 1.2 CONLAN model of time

CONLAN provides a discrete model of continuous, real time.

Real time is broken into uniform durations called "intervals" identified with integers greater than zero. Ascending, successive integers are associated with contiguous intervals. No relation between the interval and the real time second exists in general. An implementation may impose such a relation or permit users to specify such a relation.

At the beginning of each interval there are an indefinite number of computation "steps" identified with integers greater than zero. Successive steps provide a before/after relation only.

Values obtained at the last step of computation are the values associated with the interval.

When modeling a specific digital system, satisfactory results are obtained at reasonable computational cost by quantizing time to some fraction of the second; for purposes of example assume the nanosecond. Actual signals are then constrained by this quantization to change at the boundaries



of 1 ns. durations. Computing the value of a specific signal during a specific 1 ns. duration may require successive computations: if a wire is driven by a gate network modelled by  $a \wedge b \vee c \wedge d$ , then  $a \wedge b$  and  $c \wedge d$  must be evaluated before the signal value is determined. The CONLAN interval and step support this model of digital hardware and method of simulation (Figure 1).

No real time is thought to elapse when evaluating a mathematical function or executing a computer program. Yet many successive computational steps are usually required. Again the CONLAN model of time supports such computation.

## 2. FORMAL DERIVATION OF SIGNALS

In order to model real hardware components, some mechanism to describe delays in components and wires must be provided. The solution adopted in CONLAN is to keep the history of values computed at every step of every interval. Separate histories (called 'signals' in CONLAN) are kept for each component, pin, wire, etc. of the hardware system. Signals are abstractions and do not have a physical interpretation. To provide the link between the signal (i.e. a history of values) and the component, a special type of object, called a 'signal carrier' is provided by the language. In this chapter we formally define signals as a bcl type together with operations to manipulate signals. Signal carriers (carriers, for short) are the subject of the following chapter.

### 2.1 CONLAN model of computation

Hardware descriptions record how the signal parts of some carriers are related to those of other carriers. These relations display behavior and/or organization and support computation of unknown signal parts. Such computation is usually performed viewing past and present signal values as "known" and future values as "unknown." With each computational step, known values are used to determine a future value and thereby change its status to known.

The interval and step counters are managed by the envi-

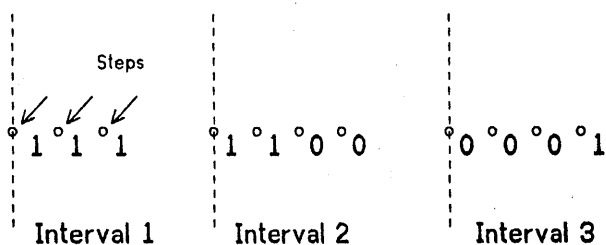


Figure 1—CONLAN model of time.

ronment. The contents of these counters are made available to toolmakers via  $t@$  and  $s@$ .

$t@$  is an integer whose value is the current time interval. Contiguous values are provided in ascending order starting with one.  $s@$  is an integer whose value is the current computation step. Contiguous values are provided in ascending order, starting with one.

When the environment determines that all signals have attained stable values, it increments the value provided by  $t@$  and resets the  $s@$  counter to 1. It detects computation step oscillation ( $s@$  reaches a predetermined limit) and responds to it with a message and optionally termination of document evaluation or continuation using the signal values available at the last step of computation.

The algorithm used by the environment is the following:

#### Stage Action

- 1 For each invoked activity and function of the system description under evaluation, determine via the definition of that invoked operation future step values from known present and past signal values. Advance to stage 2.
- 2 For all carriers which have not been serviced in stage 1, provide for them the missing step value. The determination of the missing value is the responsibility of both the environment and the toolmaker (see  $finstep@$ , below). Advance to stage 3.
- 3 Examine the record of present and next step values. If one or more signals have differing values and  $s@$  is less than a predetermined limit, advance the step counter  $s@$  and return to stage 1. If  $s@$  equals the predetermined limit, publish an "oscillation" error message and (optionally) continue with stage 4; otherwise continue with stage 4.
- 4 For each invoked activity and function of the system description under evaluation, determine the initial step value for the next interval. The determination of this step value is the responsibility of both the environment and the toolmaker (see  $finint@$ , below). Advance to stage 5.
- 5 Reset  $s@$  to 1, increment  $t@$ , and return to stage 1.

To support the model of computation, the environment uses special operations,  $finint@$  and  $finstep@$  which are provided by the toolmaker.

$Finstep@$  is an activity which describes the default signal growth mechanism for a computation step.  $Finint@$  is an activity which describes the default signal growth mechanism for a time interval.

None, one or more functions and activities may be invoked in a step for a specific carrier. If multiple invocations attempt to set a signal to different values, a "collision" exists and will be reported as an error. Operations  $finstep@$  and  $finint@$  are independent of invocations; they provide a means of providing default values or propagating values to future steps when no activities are invoked to do so.

## 2.2 Computation step signals

```

TYPE cs_signal@(x: value) BODY                                     “/value is the class of all value types/”
                                                                “/definition of the new type elements/”

ALL a: tytuple@(x) WITH size@(a) > 0 ENDALL

CARRY =, ≠, size@ ENDCARRY                                     “/imported operations from the defining type/”

FUNCTION select_css(y: cs_signal@(x), s: pint): x               “/access elements (values) in a cs_signal/”
  RETURN old@(y)[s]
  FORMAT@
  EXTEND@ ref_to_declared.5
  ref_to_declared = exp10 :id1 '{' exp7 :id2 '}'
  MEANS@ select_css (id1, id2) ENDFORMAT
ENDselect_css

FUNCTION extend_css(y: cs_signal@(x), s: pint, v: x): cs_signal@(x) “/extend a computation step signal/”
  RETURN IF s = size@(y) + 1 THEN extend(old@(y), v)
  ELIF s ≤ size@(y) THEN IF y{s} ≠ v THEN error@ ELSE y ENDIF      “/collision/”
  ELSE error@ ENDIF                                               “/order error/”
ENDextend_css
ENDcs_signal

```

A Computational Step Signal (*cs\_signal*) is the mechanism used to record a history of values during one real time interval. The definition of type *cs\_signal* indicates that the values to be recorded must all be of the same type, and the type must be specified when a *cs\_signal* is declared. Thus, one could have *cs\_signals* recording values of type integer, Boolean, etc.

The type is constructed from a more primitive type (*tytuple@*, [1]) whose elements are tuples (ordered lists) of elements of the same type. Moreover, *cs\_signals* cannot be empty ( $\text{size}@ > 0$ ) although they can be of unlimited size. For instance, the set of *cs\_signals* carrying Boolean values is:

```

{
  (. 0 .), (. 1 .),                                     “/cs_signals of length 1/”
  (. 0,0 .), (. 0,1 .), (. 1,0 .), (. 1,1 .) “/cs_signals of length 2/”
  (. 0,0,0 .), (. 0,0,1 .), (. 0,1,0 .), ... “/cs_signals of length 3/”
  .....
}

```

In addition to carrying a few operations from the defining type ( $=$ ,  $\neq$ , and  $\text{size}@$ ), *cs\_signals* provide operations for extracting or appending values to a signal.

Function *select\_css* takes two parameters (a *cs\_signal* and a position) and returns the value occupying that position in the signal:

```
RETURN old@(y)[s]
```

The value is extracted using the primitive operation *select*

(defined on elements of *tytuple@*, with infix notation  $[\dots]$ ). This operation however requires that its first parameter be an element of *tytuple@* and not an element of *cs\_signal* (or any other type). The type conversion is explicitly done by invoking a primitive operation, *old@* which takes an element of a derived type and returns the same element of the defining type.

The format statement describes an extension to the syntax. The extension is expressed in a variation of BNF in which we not only express the syntax but also the semantics of a production. In this case, the modification consists of adding one more alternatives to the definition of the non-terminal *ref\_to\_declared*. The new alternative (identified as alternative number 5) indicates that  $\{$  and  $\}$  can be used to invoke the function *select\_css*.

Function *extend\_css* takes three parameters (a *cs\_signal*, a position, and a value). It is used to compute *cs\_signals* based on an existing *cs\_signal*. Arbitrary computations of a *cs\_signal* are not performed. The *cs\_signal* returned may equal the given *cs\_signal*, or be the *cs\_signal* formed from the given *cs\_signal*, extended with the given value on the right.

Extending elements of *cs\_signal* by exactly one position ( $s = \text{size}@ + 1$ ) models the computation of step values. Attempts to extend the *cs\_signal* by more than one position violates the model of computation (see *error@*, below). A reference to a position already in use ( $s \leq \text{size}@$ ) models the simultaneous computation of values from different signal sources. Attempts to change an already recorded value ( $y\{s\} \neq v$ ) however, are reported as collision errors.

Error@ is provided by the environment as part of the system interface. When invoked, the processor of the document issues an error message. Subsequent actions are determined by the sophistication of the environment. All processing might stop; or if the nature of the error is determined, a default value may be returned and processing continued.

For example, assume a cs\_signal (a) carrying values of type integer, whose initial history is:

(. 0, -3, 1, 5 .)

Operation	Result	
select_css(a,1)	0	Extract the first recorded value.
a{4}	5	Use the syntax extension.
a{5}	error@	The signal contains only four elements.
extend_css(a,1,0)	(. 0, -3, 1, 5 .)	no change, a{1} was already 0
extend_css(a,1,1)	error@	Collision error.
extend_css(a,5,0)	(. 0, 3, 1, 5, 0 .)	Record a new value.

### 2.3 Real time signals

TYPE signals@(x: value) BODY

tytuple@(cs\_signal@(x))

CARRY =, ≠, size@ ENDCARRY

FUNCTION select\_rts(y: signal@(x), t: pint): cs\_signal@(x)

RETURN old@(y)[t]

FORMAT@ EXTEND@ ref.to.declared.5 MEANS@ select\_rts(id1, id2) ENDFORMAT

ENDselect\_rts

"/Does a value for interval t, step s exist?/"

FUNCTION known(y: signal@(x), t, s: pint): bool

RETURN size@(y) ≥ t ∧ size@(y{t}) ≥ s

ENDknown

"/Instantaneous Value/"

FUNCTION inst\_value(y: signal@(x), t, s: pint): x

ASSERT known(y, t, s) ENDASSERT

RETURN y{t}{s}

FORMAT@

EXTEND@ ref.to.declared.6

ref.to.declared = exp10 :id1 '{' exp7 :id2 ',' exp7 :id3 '}'

MEANS@ inst\_value(id1, id2, id3)

EXTEND@ ref.to.declared.7

ref.to.declared = exp10 : id1 '{' exp7 : id2 ',' '}'

MEANS@ inst\_value(id1, id2, size@(id1{id2}))

EXTEND@ ref.to.declared.8

ref.to.declared = exp10 : id1 '{' '}'

MEANS@ inst\_value(id1, size@(id1), size@(id1{size@(id1)}))

ENDFORMAT

ENDinst\_value

"/Extend a signal/"

FUNCTION extend\_rts(y: signal@(x), t, s: pint, v: x): signal@(x)

RETURN IF known(y, t, s) THEN IF y{t, s} ≠ v THEN error@ ELSE y ENDIF

"/collision/"

ELIF t = size@(y) THEN THE@ z: signal@(x) WITH@

size@(z) = size@(y) ∧

FORALL@ i: bint(1, t-1) IS@ z{i} = y{i} ENDFORALL ∧

z{t} = extend\_css(y{t}, s, v; ENDTHE

ELIF t = size@(y)+1 ∧ s = 1 THEN new@(extend(old@(y), (. v .)))

ELSE error@ ENDIF

"/order error/"

ENDextend\_rts

INTERPRETER@ FUNCTION pack@(y: x): signal@(x)

RETURN THE@ z: signal@(x) WITH@

size@(z) = t@ ∧

size@(z{t@}) = s@ ∧

FORALL@ t: bint(1, t@ - 1) IS@ z{t} = (. y .) ENDFORALL ∧

```
FORALL@ s: bint(1,s@) IS@ z{t@,s} = y ENDFORALL ENDTHE
ENDpack@
ENDsignal@
```

A Real Time Signal is the mechanism for recording interval values. During an interval, an unlimited number of computation steps may occur and these are recorded in *cs\_signals*. A signal consists of a tuple of these *cs\_signals*.

Signals are derived from type *tytuple@*. The set of signals is given by the set of all possible tuples whose elements are *cs\_signals*. For instance,

$$\text{signal@}(\text{bool}) = \{((.0.)), \dots ((.0.),(0.)), \dots ((.0,0.),(0,1.)), \dots\}$$

Function *select\_rts* takes two parameters (a signal and a position) and returns the element of the signal (a *cs\_signal*) occupying that position. This operation in fact retrieves the entire history of values computed during one interval. The formal statement takes advantage of the extension to the syntax that appeared in the definition of function *select\_css* (in type *cs\_signal@*). In the current extension, no new productions are being added, but rather the semantics of an existing alternative are extended by indicating that '{' and '}' are also used to invoke *select\_rts*. Since the parameter types are different, the type checking mechanism in the language determines which function is to be invoked.

Function *known* takes three parameters (a signal, an interval number, and a step number) and returns true or false depending on whether there is a value recorded at a given step in a given interval or not. Since *cs\_signals* do not have gaps, all that is needed is to compare the size of tuples with the interval and step numbers.

Function *inst\_value* takes three parameters (a signal, an interval number, and a step number) and returns the value recorded in that step and interval. The *ASSERT* statement monitors that the function has been properly invoked by asserting that the value is 'known'. If the assertion fails at any time during the invocation of the function, an error is reported by the interpreter. The format statements extend the language by generalizing once again the use of '{' and '}':

<i>Syntax</i>	<i>Meaning</i>
<i>css{s}</i>	Returns the value recorded during step <i>s</i> of a <i>cs_signal</i> .
<i>rts{i}</i>	Returns the <i>cs_signal</i> recorded during interval <i>i</i> of a signal.
<i>rts{i,s}</i>	Returns the value recorded during step <i>s</i> of interval <i>i</i> .
<i>rts{i,}</i>	Returns the value recorded during the last step of interval <i>i</i> .
<i>rts{ }</i>	Returns the value recorded during the last step of the last interval.

Function *extend\_rts* takes four parameters (a signal, an interval number, a step number, and a value) and computes a signal. As in the case of *cs\_signals*, only restricted computations are performed. If a value has already been recorded at the given step of the given interval, the previous value and the new value must be equal, otherwise a collision is reported by the interpreter.

If the interval referred to is the last interval of the signal,

an attempt is made to extend the *cs\_signal* recorded in that position. If the interval referred to is exactly one position beyond the last interval of the signal, and the step number is 1 (i.e. first step) then the signal is extended with a new *cs\_signal*, initialized to the new value. All other cases are excluded and reported as errors. In the expression,

$$\text{extend@}(\text{old@}(y),(v.))$$

The second parameter of *extend@* illustrates the constant denotation for tuples and derived types (*tytuple*, *cs\_signal*, *signal*).

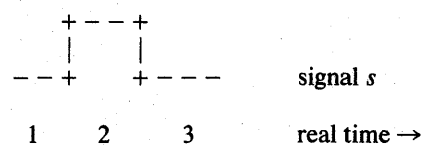
Function *pack@* is part of the system interface. By invoking this function, the environment converts a value into a signal. As in all other system interfaces which must be provided by the toolmaker, the definition of *pack@* is preceded with the keyword *INTERPRETER@*. Function *pack@* takes one parameter (a value of some type) and returns a signal capable of recording values of the same type. The signal records the history of a value that has remained constant until the current step of the current interval. This function is automatically invoked by the interpreter. It is used to provide automatic type conversion during operation invocations.

For instance, assume a Boolean signal (*s*), with the following history:

$$((.0, 0.), (.0, 1.), (.1, 1, 0.))$$

Pictorially, this is represented as:

Real Time Interval	Computation Step	Value
1	1	0
	2	0 last
2	1	0
	2	1 last
3	1	1
	2	1
	3	0 last



<i>Operation</i>	<i>Result</i>	
$s\{1\}$	(.0,0.)	The history of the first interval.
$s\{4\}$	error@	A reference to a future interval.
known( $s,1,2$ )	1	There is a value recorded at interval 1, step 2.
known( $s,1,3$ )	0	The history of interval 1 does not have three steps.
$s\{1,2\}$	0	The value at step 2 of interval 1.
$s\{1,\}$	0	The value at the last step of interval 1.
$s\{\}$	0	The value at the last step of the last interval.
extend_rts( $s,1,1,0$ )	$s$	The value at step 1 of interval 1 was already 0.
extend_rts( $s,1,1,1$ )	error@	Collision, trying to change the history.
extend_rts( $s,3,4,1$ )	((.0,0.),(.0,1.),(.1,1,0,1..))	Extended version of the current interval.
extend_rts( $s,4,1,1$ )	((.0,0.),(.0,1.),(.1,1,0.),(.1,..))	Initialize a new interval.

### 3. FORMAL DERIVATION OF CARRIERS

TYPE signal\_carrier@(x: value, di: x) BODY

cell@(signal@(x))

CARRY put@, empty@ FROM cell@ ENDCARRY

FUNCTION spart(y: carrier@(x, di)): signal@(x)

"/Signal part/"

RETURN IF empty@(y) THEN pack@(di) ELSE get@(old@(y)) ENDIF

ENDspart

"/Present signal value/"

INTERPRETER@ FUNCTION content@(y: carrier@(x, di)): x

RETURN IF empty@(y) THEN di ELSE get@(old@(y)){t@, s@}

ENDcontent

"/Real time delay/"

FUNCTION delay(y: carrier@(x, di), d: pint): x

RETURN IF  $t@ \leq d$  THEN di ELSE spart(y){t@-d, } ENDIF

FORMAT@

EXTEND@ exp10.10

exp10 = exp10 :y 'Δ' exp10 :d

MEANS@ delay(y, d)

ENDFORMAT

ENDdelay

ENDsignal\_carrier@

Type signal\_carrier@ (carrier, for short) is derived from primitive type cell@. Each cell contains a signal. Carriers are declared by specifying the type of values recorded by the signal (x) and a default/initial part (di). The role of the default/initial value is to provide a value to be used in some operations, as described below.

Function spart takes one parameter (a carrier) and returns the signal kept in the carrier. Notice the use of the default/initial value to provide a 'signal', even if no history of values has been recorded.

Function content@ is part of the system interface. It takes one parameter (a carrier) and returns the value recorded at step s@ of interval t@ of the signal (i.e. the 'current' value). It is automatically invoked by the interpreter to provide dereferencing during operation invocations.

Function delay takes two parameters (a carrier and a delay value) and returns the last value of a previous interval. Negative time is avoided by returning the default/initial value when that previous interval would be interval zero or less. The interval number is computed by subtracting the delay parameter from t@ (the current interval number). The format statement extends the language by providing an infix notation (Δ) for the delay function.

For example, assume a carrier (x) of Boolean signals with default/initial value 1. Further, assume that the history at t@ = 3, s@ = 4 is the following:

((.1,1,1.),(.1,1,0,0.),(.0,0,0,1..))

<i>Operation</i>	<i>Result</i>	<i>Signal</i>
spart(x)	((.1,1,1.),(.1,1,0,0.),(.0,0,0,1..))	Value at step s@, interval t@.
content@(x)	1	Default value.
$x \Delta 5$	1	Last value of interval 1.
$x \Delta 2$	1	

### 3.1 Terminals

```

TYPE terminal(x: value, def: x) BODY
  carrier@(x, def)
  CARRY content@, delay ENDCARRY
  ACTIVITY connect(y: terminal(x, def), a: x) BODY
    put@(old@(y), extend_rts(spart(old@(y)), t@, s@ + 1,
    a))
  FORMAT@
    EXTEND@ activity_invocation.3
    activity_invocation = ref.to.declared :id1 '='
    expression :id2
    MEANS@ connect(id1, id2)
  ENDFORMAT
ENDconnect
INTERPRETER@ ACTIVITY finstep@(y:
terminal(x,def)) BODY
  IF  $\neg$ known(spart(old@(y)), t@, s@ + 1) THEN y :=
  def ENDIF
ENDfinstep@
INTERPRETER@ ACTIVITY finint@(y:
terminal(x,def)) BODY
  put@(old@(y), extend_rts(spart(old@(y)), t@ + 1, 1,
  content@(y)))
ENDfinint
ENDterminal
SUBTYPE btm1(default: bool) BODY terminal(bool,
default) ENDbtm1
SUBTYPE btm0 BODY terminal(bool, 0) ENDbtm0
SUBTYPE btm1 BODY terminal(bool, 1) ENDbtm1

```

Activity connect has two parameters (a terminal and a value). This activity extends the (cs.signal of the current interval of the) signal associated with the terminal.

Activity finstep@ provides a terminal with a default value for the present step of the present interval if none has been provided by an invocation of connect.

Activity finint@ initializes the cs\_signal of the next interval of a terminal with the last value of the current interval.

Terminals are carriers with no retention properties. If no connect activity is invoked during a computation step to extend the signal component of a terminal, then the finstep@ activity invoked by the system extends that signal with the default value of the terminal. Activities finstep@ and finint@ force all terminal's signals to grow at the same rate.

Boolean terminals model connection points. Some connection points float high and others float low when not driven. Subtypes btm1, btm0, and btm1 are defined as shorthand for commonly used terminals.

### 3.2 Variables

```

TYPE variable(x: value, init: x) BODY
  carrier@(x, init)
  CARRY content, delay ENDCARRY
  ACTIVITY assign(y: variable(x, init), a: x) BODY
    put@(old@(y), extend_rts(spart(old@(y)), t@, s@ + 1,
    a))

```

```

FORMAT@
  EXTEND@ activity_invocation.4
  activity_invocation = ref.to.declared :id1 '='
  expression :id2
  MEANS@ assign(id1, id2)
ENDFORMAT
ENDassign
INTERPRETER@ ACTIVITY finstep@(y:
variable(x,init)) BODY
  IF  $\neg$ known(spart(old@(y)), t@, s@ + 1) THEN y :=
  content@(y) ENDIF
ENDfinstep@
INTERPRETER@ ACTIVITY finint@(y:
variable(x,init)) BODY
  put@(old@(y), extend_rts(spart(old@(y)), t@ + 1, 1,
  content@(y)))
ENDfinint@
ENDvariable

```

Variables have retention properties. They differ from terminals in the role played by function finstep@. If no assign activity is invoked during a computation step to extend the signal component of a variable, then the finstep@ activity invoked by the system extends that signal with the present value. Variables are then much as found in programming languages. Boolean variables may be thought to model abstract storage devices whose value may change at every computation step.

### 3.3 Real time variables

```

TYPE rtvariable(x: value, init: x) BODY
  carrier@(x, init)
  CARRY content, delay ENDCARRY
  ACTIVITY transfer(y: rtvariable(x, init), a: x) BODY
    put@(old@(y), extend_rts(spart(old@(y)), t@ + 1, 1,
    a))
  FORMAT@
    EXTEND@ activity_invocation.5
    activity_invocation = ref.to.declared :id1 '<-'
    expression :id2
    MEANS@ transfer(id1, id2)
  ENDFORMAT
ENDtransfer
INTERPRETER@ ACTIVITY finstep@(y:
rtvariable(x,init)) BODY
  put@(old@(y), extend_rts(spart(old@(y)), t@s@ + 1,
  content@(y)))
ENDfinstep@
INTERPRETER@ ACTIVITY finint@(y:
rtvariable(x,init)) BODY
  IF  $\neg$ known(spart(old@(y)), t@ + 1, 1) THEN y <-
  content@(y) ENDIF
ENDfinint@
ENDrtvariable

```

Real time variables model abstract storage devices whose value may change only once per real time interval. When

the transfer activity is invoked, the signal part of the real time variable is extended by one real time interval. Within an interval the computation step signal is extended with the first value—all step values are the same. A value is carried from interval to interval when no transfer is invoked.

#### 4. OTHER TYPES IN BASE CONLAN

Psc1 makes available for further use four scalar types (int, bool, string, cell@) and tuple@ as the basic structured type [1]. Using the same extension mechanisms presented in this paper, a constructor for arrays has been formally defined and will appear in a forthcoming paper. Its development follows the same pattern used in the development of signals, carriers, terminals, and variables: layers of abstraction are built by defining types, operations, and syntax extensions leading toward the final product. In contrast with the previous developments however, the concept of time steps and intervals does not play a significant role as space, rather than time is being modeled.

Due to space limitations, the full development cannot be given here. Only a sketch of the constituent types will be indicated in Figure 2.

Type tuple@ consists of ordered lists of elements of any type. Type tytuple@(t:any@) [1], consists of tuples of elements of the same type, t. A particular member of this parameterized type family is type inttuple@ (defined as tytuple@(int)). A range is defined as an inttuple of consecutive, ascending or descending integers.

An array\_dimension is a tytuple of ranges. The size of the array dimension is not limited: CONLAN supports arrays of any number of dimensions. An element\_index is an inttuple. It describes, with respect to a particular array\_dimension the elements along each dimension needed to access a single element of an array. A slice\_index is a tytuple@(range). It describes, with respect to a particular array\_dimension, the ranges of elements along each dimension needed to access a slice of an array.

Type array(d: array\_dimension@, t:any@) is defined as a tuple of two elements. The dimensions part (of type array\_dimension@) describes the dimensions of the array; the value part (of type tytuple@(t)) is the list of array elements.

An abbreviated list of operations defined on arrays is de-

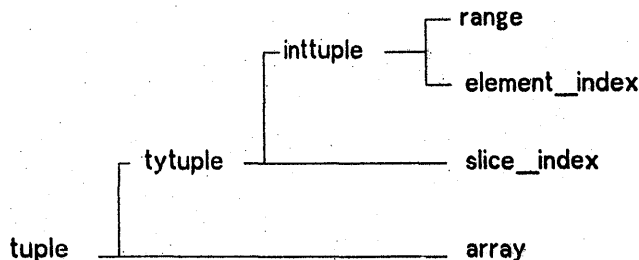


Figure 2—Development of arrays.

picted below:

```

select_element(x:array(d,t), z: element_index): t
  selects an element of an array.
select_slice(x:array(d,t), z: slice_index): t
  selects a slice of an array.
compatible(x, y: array(d,t)): bool
  tests if two arrays have compatible dimensions
equal(x, y: array(d,t)): bool
  tests if two arrays have identical value parts.
eq(x, y: array(d,t)):array(d,bool)
  tests if corresponding elements of compatible arrays
  are equal.
  
```

For many of these operations and types, special infix formats and constant denotation formats have been defined via the syntax extension mechanism.

#### 5. CONCLUSIONS

In this paper we have presented the basic mechanisms used in CONLAN to define and extend languages. The mechanism is based not only on the definition of new types and operations but also on the extension of the syntax of a base language.

The concept of the interpreter is basic to the family of languages. The interpreter provides the basic counters used to model elapsed time (t@ and s@) and the detection of error conditions (error@). It can also be augmented in a controlled manner by the toolmakers. This is achieved by the definition of special functions and activities which can then be invoked automatically by the interpreter. In this paper we have shown a few of these operations (pack@, content@, finint@ and finstep@) which are used to provide restricted dereferencing and signal growth mechanisms.

A full coverage of the base language is outside the size limitations of this paper. Enough has been presented however, to motivate the reader to appreciate the power of the notation and its usefulness in the development of a comprehensive family of languages for describing the behavior and interconnection of hardware components.

#### 6. ACKNOWLEDGMENTS

The authors are indebted to Bell Northern Research (Ottawa), Sperry Univac (Philadelphia), Office of Naval Research, Ballistic Missile Defense Advanced Technical Center (Huntsville), IRIA (Paris), Bundesministerium fur Forschung und Technologie (Bonn), Siemens (Munich), and Fujitsu (Tokyo) for their interest and support, Professor Yaohan Chu for his early contributions, and in particular to Professor Jack Lipovski for his help and unwavering confidence in the group.

---

## 7. REFERENCES

1. Piloty, R., Barbacci, M., Borrione, D., Dietmeyer, D., Hill, F., and Skelly, P., "CONLAN—A Formal Construction Method for Hardware Description Languages: Basic Principles," *Proceedings National Computer Conference*, Volume 49, Anaheim, California, 1980.
2. Piloty, R., Barbacci, M., Borrione, D., Dietmeyer, D., Hill, F., and Skelly, P., "CONLAN—A Formal Construction Method for Hardware Description Languages: Language Application," *Proceedings National Computer Conference*, Volume 49, Anaheim, California, 1980.





# CONLAN—A formal construction method for hardware description languages: language application

by ROBERT PILOTY

*Technische Hochschule Darmstadt, FR Germany*

MARIO BARBACCI

*Carnegie-Mellon University*

DOMINIQUE BORRIONE

*Universite de Grenoble, France*

DONALD DIETMEYER

*University of Wisconsin-Madison*

FREDRICK HILL

*University of Arizona-Tucson*

and

PATRICK SKELLY

*Honeywell, Phoenix*

## 1. INTRODUCTION

CONLAN is a formal semantic and syntactic base for the description of all phases in the design and documentation of digital systems. Previous papers[1,2] have presented the basic concepts and models underlying the whole CONLAN language family. On the example of the construction of Base CONLAN, the first level which can be of some operational interest to the hardware designer, it has been shown how a coherent set of special purpose application languages can be derived from a common mathematical base, Primitive Set CONLAN.

This paper emphasizes the logic system designer's point of view. By this, we mean the user of a CONLAN language rather than the definer of a member of the CONLAN language family. The reader is assumed to know the available object types and operations in Base CONLAN, which will be, for the purpose of this paper, considered as a user language.

The authors are convinced that the CONLAN concepts are general and versatile enough to be applicable to several design methodologies. In particular, the necessity of being able to decompose a system into modules is now widely recognized. The first part of this paper presents the CONLAN basic user's segmentation mechanism, called DESCRIPTION. Subsequent parts show how the same "hardware object" can be either considered to be a DESCRIPTION, an ACTIVITY or an object TYPE, depending upon the level of abstraction of its model written in CONLAN.

## 2. DESCRIPTION SEGMENT

Any piece of CONLAN text which models a system or a subsystem is written as a DESCRIPTION segment. A DESCRIPTION can represent as small a piece of hardware as a NAND gate, or as big a system as a computer network. The main characteristic of a DESCRIPTION segment is that it is considered as a module in itself, independently of whatever environment it may be inserted in; the best visualization of a DESCRIPTION segment is an integrated circuit.

A DESCRIPTION segment knows only its interface carriers, internal carriers and operators, and eventually subsystems into which it is further decomposed. It has no knowledge of global carriers, and all communications with its environment are done through its interface.

Several subsystems may be very similar in function and structure, except for some very specific parameter. For instance, an 8 bit and a 16 bit parallel adder perform the same function, and are of the same nature, except for the dimensions of their operands. CONLAN does not require the user to define two distinct DESCRIPTION segments in such a case, but provides a facility to indicate ATTRIBUTE parameters. A DESCRIPTION segment definition with attributes therefore describes a family of modules. From that family, instances with specific attribute values will be used in a particular context.

The reference language for all segments in this paper is taken to be Base CONLAN (bcl), as illustrated in the first example.

2.1 Structure of a DESCRIPTION segment

The minimum amount of information a DESCRIPTION segment should contain is:

1. the name of the segment
2. the list of its interface carriers, specifying their type and direction (IN for input, OUT for output, INOUT for bidirectional)
3. operation invocations showing how the signals of the OUT and INOUT carriers are related to the signals of the IN, INOUT, and internal carriers.

A very simple example is a 2 input, 1 output unit delay nand gate:

```
REFLAN bcl
DESCRIPTION nandgate1(IN x,y: btm1, OUT z: btm1)
BODY
z . = (x  $\neg$   $\wedge$  y)  $\Delta$  1
ENDnandgate1
```

where  $\Delta$  is the delay operator,  $. =$  denotes terminal connection, and btm1 has been defined as the type "Boolean terminal with default value 1" [2].

One could wish to make the value of the delay an attribute of the DESCRIPTION, so as to write once and for all the model of a 2 input - 1 output nand gate with any propagation delay. In this case, the interface list is augmented with an ATTRIBUTE section containing the delay parameter d, which must be fixed at compile time.

```
DESCRIPTION nandgate(ATT d: pint, IN x, y: btm1,
OUT z: btm1)BODY
z . = (x  $\neg$   $\wedge$  y)  $\Delta$  d
ENDnandgate
```

More complex descriptions can be written, in terms of an interconnection of instances of nand gates. This is shown on the following model of the 3 input, 1 output multiplexer displayed in Figure 1.

```
DESCRIPTION multiplex (IN a, b, c: btm1, OUT d:
btm1)BODY
DESCRIPTION nandgate1 (IN x, y: btm1, OUT z:
btm1)
BODY z . = (x  $\neg$   $\wedge$  y)  $\Delta$  1 ENDnandgate1
DECLARE i, j, k: btm1 ENDDECLARE
USE g1, g2, g3, g4: nandgate1 ENDUSE
```

```
g1.x = a,   g1.y = a,   i = g1.z,
g2.x = b,   g2.y = a,   j = g2.z,
g3.x = i,   g3.y = c,   k = g3.z,
g4.x = j,   g4.y = k,   d = g4.z
ENDmultiplex
```

Four distinct, explicitly named gates are specified, in the USE statement, as instances of nandgate1. Local terminals

i, j, k are declared. The interface terminals of the individual gates are referenced through dot notation, and individually connected to interface and local carriers of multiplex in the operation invocation part of that description.

We now have illustrated the general constituents of a DESCRIPTION segment, which is divided into 5 parts, of which only the first one must not be empty.

1. Description header, consisting of  
DESCRIPTION  
description name  
(list of attributes and interface carriers)  
BODY

2. Assertions part, consisting of  
ASSERT list of predicates ENDASSERT

The predicates may be static conditions on attribute parameters, to ensure that the model is meaningful. They may also express dynamic constraints (functional and time dependencies) on interface and/or internal signals. All predicates must be true at every point in time during the simulation of an instance of a DESCRIPTION segment.

3. Definition part, defining operation and description segments to be used for portraying the behavior and structure of the description being defined. These definitions may be locally written; or they may be brought from an existing library of segments, in which case they are said to be EXTERNAL, and only their name and parameter/interface are written.

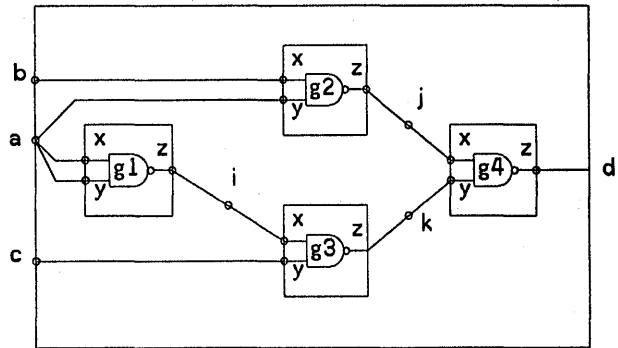


Figure 1(a)—Interconnection as operation invocation.

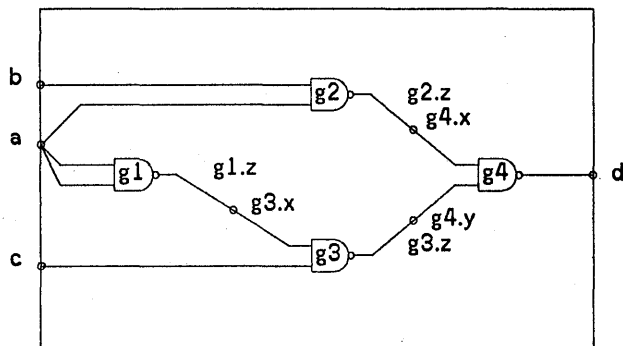


Figure 1(b)—Permanent interconnection.

4. Internal objects, where local carriers are named and typed in a DECLARE statement. If the description is constructed from smaller descriptions (which must then be defined in part 3), instances of those descriptions are named in a USE statement.
5. Operation invocation part, describing the input/output behavior of the hardware unit. It terminates with the usual END keyword, possibly followed by the description name.

### 3. INSTANTIATION AND INTERCOMMUNICATION

Descriptions may be instantiated and interconnected in an enclosing description in either of two different ways.

#### 3.1 Intercommunication in the operation invocation part

The first method was shown on the multiplex description above. Instance names are brought to existence in the USE statement, and actual attributes (if any) are given at that point.

This implicitly declares all the interface carriers of the instances, which will be referred to, in the enclosing description, by compounding the instance name with the formal interface carrier names.

Intercommunications between enclosed instances of descriptions, and communications between instances and the enclosing description, are stated in the operation invocation part of the enclosing description.

If all interface carriers of enclosed instances are terminals, and all these terminals are interconnected, then an explicit wiring list is being written, as in multiplex. In that case, intermediate carriers *i,j,k* are not necessary. The nandgate description, previously defined and assumed in the library, is used to show the EXTERNAL and instantiations with actual attributes in the equivalent multiplex2 description below:

```
DESCRIPTION multiplex2 (IN a, b, c: btm1, OUT d:
btm1) BODY
EXTERNAL DESCRIPTION nandgate
ENDEXTERNAL
USE g1, g2, g3, g4: nandgate (ATT 1) ENDUSE
g1.x . = a,    g1.y . = a,    d . = g4.z,
g2.x . = b,    g2.y . = a,
g3.x . = g1.z, g3.y . = c,
g4.x . = g2.z, g4.y . = g3.z
ENDmultiplex2
```

At some more abstract level of a design decomposition of a description might be more conceptual than physical. Then, interface carriers may be of types other than terminal, like registers and variables for instance. And communications to enclosed instances of descriptions would be done through other operations: register transfers, variable assignments and the like. In a more abstract model, communications could even be under the scope of conditions, with some hard-

ware implied, rather than actually defined. For instance, in the structure of Figure 2, one could wish to consider registers *a* and *b* to be interface carriers of the alu, and to portray the behavior rather than the structure of the local memory and control part.

A very simplified model is shown below:

```
DESCRIPTION structure BODY
DESCRIPTION alunit (IN a[0:7], b[0:7]: brtv(0),
ctl[0:3]: btm0, OUT r[0:7], cnd[0:1]: btm0),
...
ENDalunit
...
DECLARE local_memory [0:7, 0:15]: brtv(0),
ad: tml(int,0),
ca, cb, cm: btm0,
...
ENDDECLARE

USE alu: alunit ENDUSE
...
IF ca THEN alu.a <- local_memory [,ad] ENDIF,
IF cb THEN alu.b <- local_memory [,ad] ENDIF,
IF cm THEN local_memory [,ad] <- alu.r ENDIF,
...
ENDstructure
```

where  $\leftarrow$  denotes rtvariable transfer, *btm0* has been defined as the type "Boolean terminal with default value 0," and *brtv* has been defined as the type "Boolean real time variable" [2]. The rtvariable transfer is accomplished in one unit of real time.

#### 3.2 Permanent intercommunication

The second method by which an instance of a description can communicate with its environment is by considering that its interface carriers permanently drive or are driven from carriers of the enclosing description or interface carriers of other instances. This is indicated in the USE statement: the instance name is followed by the list of the environment carriers which the implicitly declared interface carriers drive or are driven from.

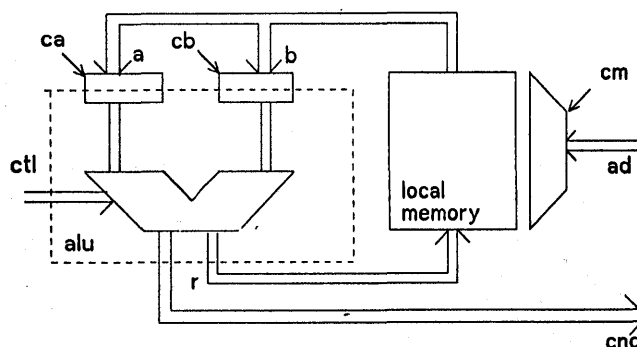


Figure 2

```

DESCRIPTION multiplex3 (IN a, b, c: btm1, OUT d:
btm1) BODY
EXTERNAL DESCRIPTION nandgate
ENDEXTERNAL
USE g1 (IN a, a, OUT g3.x),
   g2 (IN b, a, OUT g4.x),
   g3 (IN g1.z, c, OUT g4.y),
   g4 (IN g2.z, g3.z, OUT d): nandgate (ATT 1)
ENDUSE
ENDmultiplex3

```

The above example is equivalent to multiplex2, previously studied.

#### 4. ABSTRACTION LEVELS IN CONLAN MODELS

DESCRIPTION segments provide the user with a structuring means to describe how a digital system is constructed from a set of components (network description). ACTIVITY and FUNCTION segments on hand, give him the way to express what signal state changes occur in the carriers of the digital system at every given point in time (behavior description). CONLAN does not go down to the electronic component level of a description; the most detailed description the user can write is therefore a logic gate network. The hardware designer however is not necessarily interested in gate-level details. He might not even know, at some stage of a design, neither how many components he will be actually using, nor which specific ones, in some part of his system.

CONLAN provides maximum flexibility for stepwise refinement of a design. Instances of purely behavioral descriptions can be interconnected with instances of detailed network descriptions. Moreover, inside the same DESCRIPTION segment, some parts may be expressed in terms of clearly identified hardware elements, while other parts might be abstract.

Our purpose in this section is to show how the same physical object may be more or less precisely specified, depending upon the CONLAN construct for which it stands.

##### 4.1 Duality between TYPE and DESCRIPTION

Internal objects which are named in a description, and which therefore can be inspected for state changes, are of two categories: (a) carriers, which are elements of types and, (b) instances of description.

Although the user will probably not often define new types, he may select (or ask for) a language where some types are primitive, or decide to construct an object as a description in terms of more elementary objects.

##### Flip-flops as description segments

For instance, Base CONLAN has neither primitive flip-flop nor register types. But these can be constructed as DESCRIPTION segments. Assuming unit delay gates, the network description of a rising edge triggered D-flip-flop would

read:

```

DESCRIPTION dff1 (IN c,d: btm1 OUT q,nq: btm0)
BODY
DESCRIPTION nand2 (IN x,y: btm1 OUT z: btm0)
BODY
z . = x Δ 1 ¬∧ y Δ 1
ENDnand2
DESCRIPTION nand3 (IN w,x,y: btm1 OUT z: btm0)
BODY
z . = ¬(w Δ 1 ∧ x Δ 1 ∧ y Δ 1)
ENDnand3
USE g1, g2, g4, g5, g6: nand2, g3: nand3 ENDUSE
g1.x . = g4.z,   g1.y . = g2.z,
g2.x . = g1.z,   g2.y . = c
g3.w . = g2.z,   g3.x . = c,   g3.y . = g4.z,
g4.x . = g3.z,   g4.y . = d,
g5.x . = g2.z,   g5.y . = g6.z,
g6.x . = g5.z,   g6.y . = g3.z,
q . = g5.z,      nq . = g6.z
ENDdff1

```

The wiring of the NAND gates is clear enough. Since all gate delays and initial values are identical, real time oscillation can be expected initially.

While the following description may be thought to provide the same internal organization as the description above, it essentially provides a signal computation model of the D-flip-flop.

```

DESCRIPTION dff2 (IN c,d: btm1 OUT q,nq: btm0)
BODY
DECLARE e, f, g, h: btm1 ENDDECLARE
e . = h Δ 1 ¬∧ f Δ 1,
f . = e Δ 1 ¬∧ c Δ 1,
g . = ¬(f Δ 1 ∧ c Δ 1 ∧ h Δ 1),
h . = g Δ 1 ¬∧ d Δ 1,
q . = f Δ 1 ¬∧ nq Δ 1,
nq . = q Δ 1 ¬∧ g Δ 1
ENDdff2

```

At some more abstract level, we want to think of a flip-flop as a 2 state device. Then, correct initial values are assumed inside the module. Oscillations due to default in initialization or wrong setting of data and clock inputs, can no longer be displayed on such a simplified model. However, it is possible to ASSERT properties on input signals to ensure that the description is reacting correctly when the environment produces acceptable waveforms. The following description provides a behavioral 3 units of time delay model of the d-flip-flop:

```

DESCRIPTION dff3 (IN c,d: btm1, OUT q: brtv(0), nq:
brtv(1)) BODY
ASSERT ¬ c ∨ d = d Δ 1) ENDASSERT
"/we assert stability of d when c is 1"/
IF ¬ c Δ 4 ∧ c Δ 3 THEN q <- d Δ 3, nq <- ¬ d
Δ 3 ENDIF
ENDdff3

```

The IF condition detects the rising edge of the *c* input. The transfers introduce a 3 unit propagation delay, to represent 3 layers of unit delay nand gates. The two values of *q* provide the two states we desired. *nq* has been kept only for interface equivalence with previous descriptions. Other varieties of flip-flops may be modeled similarly to the above 3 models. Just one last behavioral example is provided, for a master-slave flip-flop. In the ASSERT statement, we insist that *j* and *k* do not change value when the clock is high. A 4 unit propagation delay stands for the 4 layers of nand gates.

```
DESCRIPTION jkff3(IN c,j,k: btm0,OUT q: brtv(0),
nq: brtv(1)) BODY
ASSERT  $\neg c \vee (j = j \Delta 1 \wedge k = k \Delta 1)$ 
ENDASSERT
IF  $\neg c \Delta 5 \wedge c \Delta 4$  THEN
  IF  $j \Delta 4 \neq k \Delta 4$  THEN  $q \leftarrow j \Delta 4, nq \leftarrow k \Delta 4$ 
  ELIF  $j \Delta 4$  THEN  $q \leftarrow nq, nq \leftarrow q$  ENDIF
ENDIF
ENDjkff3
```

Instances of such descriptions may now be used to represent the existence of flip-flops and registers in a circuit as shown in Figure 3, where a register is depicted as an array of flip-flops.

```
DECLARE fa, fb: btm0 ENDDECLARE
USE a[0:15], b[0:15], r[0:15]: dff2 ENDUSE
OVER i FROM 0 TO 15 REPEAT
  r[i].c := fa  $\vee$  fb,
  r[i].d := a[i].q  $\wedge$  fa  $\vee$  b[i].q  $\wedge$  fb
ENDOVER
```

Here, the OVER statement has been used to compactly write repetitive connections. A macro-generation into 16 pairs of connections is expected from the software.

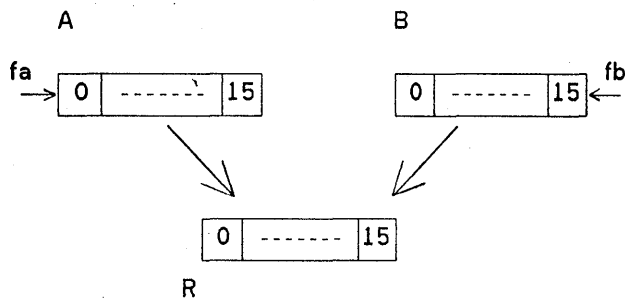


Figure 3(a)—Global structure.

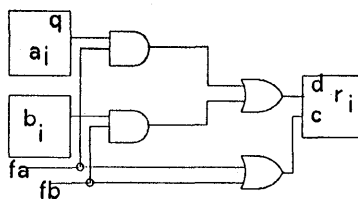


Figure 3(b)—Detail of routing.

### Flip-flops as carrier types

In register transfer level hardware modeling, emphasis is no longer on data paths between memory elements, but rather on state changes. Taking this point of view, the hardware designer sees flip-flops and registers as carriers rather than descriptions, and portrays state transitions by explicit operations rather than by signal changes on interfaces. The following type segment illustrates the definition of a *jk*-flip-flop as it would appear in a language definition [1]. The three activities provide for the three combinations of *jk* input values which change the flip-flop state. Unit delay is assumed for convenience.

```
TYPE jkff BODY brtv(0)
CARRY content@, delay brtv(0) ENDCARRY
ACTIVITY set (c: btm0, x: jkff) BODY
  "/j = 1, k = 0/"
IF  $\neg c \Delta 1 \wedge c$  THEN old@(x)  $\leftarrow$  1 ENDIF
ENDset

ACTIVITY reset (c: btm0, x: jkff) BODY
  "/j = 0, k = 1/"
IF  $\neg c \Delta 1 \wedge c$  THEN old@(x)  $\leftarrow$  0 ENDIF
ENDreset

ACTIVITY toggle (c: btm0, x: jkff) BODY
  "/j = k = 1/"
IF  $\neg c \Delta 1 \wedge c$  THEN old@(x)  $\leftarrow$   $\neg$  x ENDIF
ENDtoggle
ENDjkff
```

At this point, a one to one correspondence is still present between the carrier type *jkff* and the hardware component. A conditional transfer such as Figure 3.b could be embedded in a description by:

```
DECLARE a, b, r: jkff c, fa, fb: btm0 ENDDECLARE
c := fa  $\neg$  fb,
IF fa THEN IF a = 1 THEN set(c,r) ELSE reset(c,r)
ENDIF ENDIF,
IF fb THEN IF b = 1 THEN set(c,r) ELSE reset(c,r)
ENDIF ENDIF
```

More abstraction is obtained in the following *ff* type definition segment, where only one activity load, with an additional parameter, replaces set, reset, and toggle. The *ff* carrier type represents both, and allows more compact descriptions, with a loss in precision with respect to the actual component it is supposed to model:

```
TYPE ff BODY brtv(0)
CARRY content@, delay ENDCARRY
ACTIVITY load (c: btm0, d: bool, x: ff) BODY
IF  $\neg c \Delta 1 \wedge c$  THEN old@(x)  $\leftarrow$  d ENDIF
ENDload
ENDff
```

If TYPE *ff* were to be given as a primitive to the user of

a CONLAN language, a FORMAT statement would probably be attached to ACTIVITY load, in order to define a more convenient infix notation, such as

$$?c? x \leq d$$

But such possibilities of abbreviation are not our point in this paper, and we shall retain the standard prefix notation.

If  $y$  has been declared ff, the statement:

```
IF ci THEN load (c, ¬y, y)
```

can be implemented by either of the circuits in Figure 4.

The module 6 counter of Figure 5, whether implemented with one type of flip-flop or another, is described by:

```
DECLARE r1, r2, r3: ff, c1: btm0 ENDDECLARE
load(c1, ¬r3, r1),
load(c1, r1, r2),
load(c1, r2, r3)
```

Using this abstract carrier type ff, the conditional transfer of Figure 3 can be simply written as:

```
DECLARE fa,fb,c: btm0, a[0:15],b[0:15],r[0:15]: ff
ENDDECLARE
c . = fa ∨ fb,
OVER i FROM 0 to 15 REPEAT
  IF fa THEN load (c, a[i], r[i]) ENDIF,
  IF fb THEN load (c, b[i], r[i]) ENDIF
ENDOVER
```

#### 4.2 Duality between operation and DESCRIPTION

In a digital system, some components are viewed primarily as operations which transform input signals, seen as operands, to produce new values. Typical examples are combinatorial circuits, but sequential operative modules such as microprocessors are not excluded.

CONLAN supports two ways of representing such components. If the hardware designer is mainly interested in describing behavior, he will define and invoke FUNCTION and ACTIVITY segments. If structure must be displayed, he will define and instantiate DESCRIPTION segments. The choice is not dictated by the level of detail he wishes to display: all of these segments can be used for various levels of abstraction. Yet a fundamental difference exists as to how much hardware is implied in the designers text. If the same FUNCTION or ACTIVITY segment is invoked three times

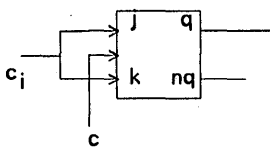


Figure 4(a)—y as JKFF.

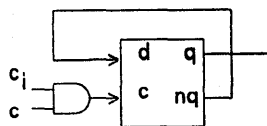
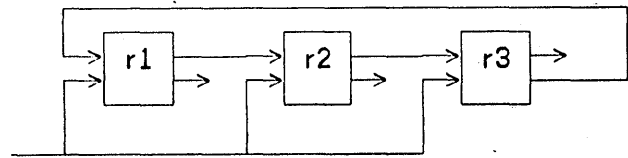


Figure 4(b)—y as dff.



c1

Figure 5—Module 6 counter.

in a model, there is no a priori indication of the number of hardware units: there could be one (with multiplexing or time sharing) or as many as three. On the contrary, if a DESCRIPTION segment is used, there are exactly as many distinct hardware components as are instantiated. And in fact, in a top-down approach, going from a model expressed in terms of operations to a model expressed in terms of instances of descriptions is precisely what is called synthesis.

This duality between operation and instance of description was suggested in the description of flip-flops dff1 and dff2 presented above. We shall illustrate it here with additional examples.

#### An adder as FUNCTION and as DESCRIPTION

Suppose we have defined the two types: (a) *btmvector* to be a normalized (1:n) vector of Boolean terminals, default 0, and (b) *boolvector* to be a normalized vector of Booleans.

A generalized parallel adder, with interface carriers being Boolean terminals, can be written for all possible lengths of its  $x$  and  $y$  inputs, and  $z$  result, provided the size of  $x$ ,  $y$  and  $z$  are equal:

```
DESCRIPTION adder(IN x,y: btmvector, cin: btm0
  OUT z: btmvector, cout: btm0) BODY
ASSERT size = size(y), size(z) = size(y) ENDASSERT
DECLARE c[0:size(x)]: btm0 ENDDECLARE
```

```
c[size(x)] . = cin,
OVER i FROM 1 TO size(x) REPEAT
  z[i] . = x[i] XOR y[i] XOR c[i],
  c[i-1] . = x[i] ∧ y[i] ∨ x[i] ∧ c[i] ∨ y[i] ∧ c[i]
ENDOVER,
cout . = c[0]
ENDadder
```

On the other hand, an adder may be viewed as a function on two Boolean vectors of same size, returning a Boolean vector with one more element, the leftmost bit being the carry.

```
FUNCTION add (x, y: boolvector, cin: bool):
  boolvector BODY
ASSERT size(x) = size(y) ENDASSERT
DECLARE c[0:size(x)], z[1:size(x)]: btm0
ENDDECLARE
```

```

c[size(x)] . = cin,
OVER i FROM 1 TO size(x) REPEAT
  z[i] . = x[i] XOR y[i] XOR c[i],
  c[i-1] . = x[i] ^ y[i] v x[i] ^ c[i] v y[i] ^ c[i]
ENDOVER
RETURN c[0] # z      /* # is catenate operator */
ENDadd

```

At the point of use of either segment, the size of their inputs must be known.

Suppose we want to perform addition on two bytes, and the model includes the following DECLARE statement:

```

DECLARE a[1:8], b[1:8], s[1:8], ca: btm0
ENDDECLARE

```

A number of possible statements will produce the same effect in terminal s and ca:

```

USE addition(a, b, 0, s, ca): adder ENDUSE
USE addition1(a[5:8], b[5:8], 0, s[5,8], addition 2.cin),
  addition2(a[1:4], b[1:4], addition 1.cout, s[1:4],ca):
  adder
ENDUSE
ca # s . = add(a, b, 0)

```

The first method clearly identifies one 8 bit adder; the second method shows two 4 bit adders; the third method does not call for a particular hardware implementation.

#### An alu as ACTIVITY and as DESCRIPTION

From IC data books, the logic diagrams and truth tables for each component can be modeled by a CONLAN description segment. If the hardware designer had access to a library of such descriptions, the logic of a CONLAN description could be tested, simulated, and verified by software.

For instance, a 4 bit MSI alu chip is represented below. Only logic functions are spelled out, for reasons of space. A one to one correspondence exists between the actual IC pins and the DESCRIPTION segment interface carriers, except for power supply pins which are not taken into account.

```

DESCRIPTION mc10181(IN a[0:3], b[0:3], s[0:3], cin,
m: btm0, OUT f[0:3], co, pg, gg: btm0) BODY
If m = 1 THEN      /* logic functions */
  CASE s IS
    0000: f . = ~ a,
    0001: f . = ~ a v ~ b,
    0010: f . = ~ a v b,
    0011: f . = 1111,
    0100: f . = ~ a ^ ~ b,
    0101: f . = ~ b,
    0110: f . = a XOR b,
    0111: f . = a v ~ b,
    1000: f . = ~ a ^ b,

```

```

1001: f . = a EQV b,
1010: f . = b,
1011: f . = a v b,
1100: f . = 0000
1101: f . = a ^ ~ b,
1110: f . = a ^ b,
1111: f . = a
ENDCASE

```

```

ELSE ...          /* arithmetic functions */
ENDIF
ENDmc10181

```

The same circuit can as well be written as an ACTIVITY segment. To do so, it is only sufficient to change the header, the body can be identical. However, for more clarity, input parameters can be typed bool, rather than Boolean terminal, to show that this ACTIVITY segment treats them as values.

```

ACTIVITY mc10181a(a[0:3], b[0:3], s[0:3], cin, m: bool,
  f[0:3], co, pg, gg: btm0) BODY
.....          /*same as above*/
ENDmc10181a

```

Usage of one formulation or the other will be dictated, once again, by whether or not the actual number of ICs, and their exact interconnection, is to be displayed.

Constructing a 32 bit alu from 4 bit slices would imply instantiating 8 mc10181 DESCRIPTION segments. Whereas the two conditional invocations

```

IF c1 THEN mc181a (.....) ENDIF
IF c2 THEN mc181a (.....) ENDIF

```

might imply multiplexing, if c1 and c2 are mutually exclusive.

## 5. CONCLUSIONS

Base CONLAN is primarily a starting point, with well defined and semantically sound primitives, for language designers, to derive a coherent and comprehensive family of digital system description languages. As a result, writing hardware descriptions in Base CONLAN, although quite possible as we have seen, may look verbose. More time and effort will be needed before user oriented, less general but simpler, special purpose languages will be developed.

Nevertheless, all concepts pertinent to hardware modeling are already here, and will be common to all languages of the CONLAN family. The ability to partition a design into a network of interconnected modules is supported by the notion of DESCRIPTION segments. Instantiation of such segments, and intercommunication between instances, directly depicts the structure of a system, and the data and control paths which connect its components.

Declaring objects of a carrier TYPE may correspond to a physical hardware unit, but may also constitute an abstraction. In that sense, it is expected that hardware com-



riters, fed with alternative libraries of DESCRIPTION segments, will generate models specialized for a given technology. CONLAN also provides FUNCTION and ACTIVITY segments, to describe the behavior of a system, often in more concise manner than when structure is fully displayed.

Space limitations prevented us from giving other than small examples. However, we hope to have made clear to the reader that, at each stage of the design of a real project, judicious selection of CONLAN segments leads to descriptions which are both adapted to the available amount of information and writing style.

## 6. ACKNOWLEDGMENTS

The authors are indebted to Bell Northern Research (Ottawa), Sperry Univac (Philadelphia), Office of Naval Re-

search, Ballistic Missile Defense Advanced Technical Center (Huntsville), IRIA (Paris), Bundesministerium für Forschung und Technologie (Bonn), Siemens (Munich), and Fujitsu (Tokyo) for their interest and support, Professor Yaohan Chu for his early contributions, and in particular to Professor Jack Lipovski for his help and unwavering confidence in the group.

## 7. REFERENCES

1. Piloty, R., Barbacci, M., Borrione, D., Dietmeyer, D., Hill, F. and Skelly, P., "CONLAN—A Formal Construction Method for Hardware Description Languages: Basic Principles," *Proceedings National Computer Conference*, Volume 49, Anaheim, California, 1980.
2. Piloty, R., Barbacci, M., Borrione, D., Dietmeyer, D., Hill, F. and Skelly, P., "CONLAN—A Formal Construction Method for Hardware Description Languages: Language Derivation," *Proceedings National Computer Conference*, Volume 49, Anaheim, California, 1980.

# Design decisions for the intelligent database machine

by ROBERT EPSTEIN and PAULA HAWTHORN

*Britton-Lee Inc.*  
Albany, California

## 1. INTRODUCTION

The Intelligent Database Machine (IDM), manufactured by Britton-Lee Inc., is a back-end processor and storage system that contains a complete data management system. It includes specialized hardware to perform data management functions. It is our contention that the designers of a database machine must choose a focus for their machine which strongly influences other design decisions. The IDM is low cost, high performance machine designed to support "mid-range" users. This paper presents the reasons for this choice, and the resultant design issues.

The discussion is divided into four sections. In section two we describe our conception of the population of database users. We show that a single machine cannot solve all classes of user performance and cost problems. We identify the class of applications for which we focus the IDM. In section three we discuss the design trade-offs for that class of applications. The last section, section 4, is the conclusion.

## 2. TARGET USER POPULATION

There are four user attributes which affect the design of a database machine. These are:

(1) *Required transaction rate*

If an extremely high transaction rate is required, a high degree of parallelism and/or very fast storage (semiconductor memory, fixed head disks), must be incorporated into the design of the database machine.

(2) *Storage requirements*

Very large storage requirements preclude the use of expensive storage as the total storage required, and mandate a multi-level storage system.

(3) *Wealth*

The cost of the database machine is the major constraint in its design.

(4) *Predictability of access to data*

Is the data base generally accessed in a predictable manner? Certain applications naturally reference data by one or more keyed values, for example, part numbers, employee numbers, employee names, etc. In some applications, there may be many possible access patterns, for example, census data and other statistical

applications. Database machines can be designed toward the expected usage pattern.

Using these four attributes we shall attempt to define general user populations. Let us define four general user categories: the small business system, the scientific and medium business system, the large business system, and the special purpose system. Figure 1 shows the transaction rates and storage requirements for these categories of users. The transaction rate is on the horizontal axis, and the storage requirements on the vertical axis. A small business system has relatively few users and correspondingly a low transaction rate. The storage requirements for the small business system are also relatively small.

The transaction rate and storage requirements increase proportionately for the medium-scale and scientific systems, the large business systems, and, finally, the special purpose system, where the database is huge and/or the necessary transaction rate very high. Of course, there will be exceptions to this conceptual graph: there are systems that are small and require very fast access, or large and have relatively low transaction rates. However, most systems appear to fall in the general categories marked on the graph.

The cost of a system must be related to the performance needed by the end user. Those who require high-performance, large storage systems can usually pay premium prices for them; the small to medium scale users are usually the

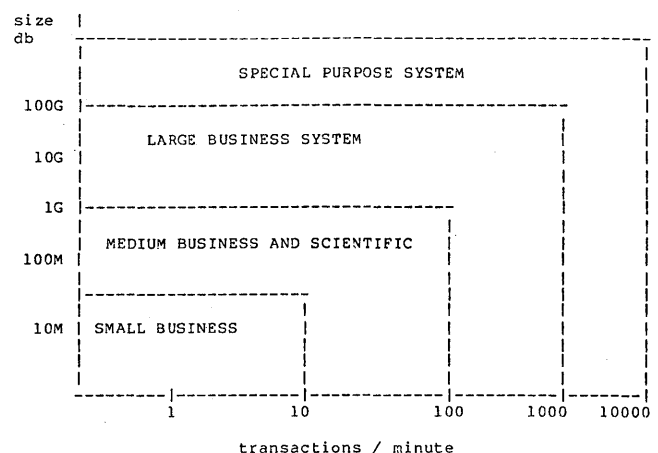


Figure 1—User classification.

ones who do not need, and do not wish to pay for, extremely high performance.

Unlike attributes (1) through (3), predictability of access cannot be correlated with the size of the database. It is an attribute of many general purpose business database applications. A typical example is a business system, where there is an employee file. If the file contains employee name, address, phone number, employee number, etc., it is very likely that it will be accessed by specifying a name or employee number, and very unlikely that it will be accessed by specifying a phone number. This means it is reasonable to expect a request such as "What is Smith's phone number?" and not "Who has phone number 527-7646?"

We shall now show that the choice of target user population dictates the design of the machine.

### 2.1. Possible decisions

If a database machine is designed for users who need extremely high transaction rates it must include either very fast storage (ram or fixed head disks), as in DIRECT [DEWI79], or it must include a mechanism to parallel search the disk, as in CASSM [LIPO78]. On the other hand, if the focus of the machine is toward the user of very small databases, under 1M on our graph, and a low transaction rate, a stand-alone microprocessor system might suffice. The focus of the machine dictates the design of the machine because each increase in performance (either storage or transaction rate) results in an increase in price. As the price goes up, the potential market changes.

### 2.2. The IDM decision

It was decided to focus the IDM toward the mid-scale user. By "mid-scale" we mean users who require transaction rates in the range of 100 to 1000 per minute (possibly higher in certain applications). We further expect mid-scale users to have applications where the majority of transactions have predictable access patterns. This decision was made because large and special users are well served by other database machines being developed, such as DBC [BAUM76], MUFFIN [STON79] and DIRECT [DEWI79]. Small to medium scale systems are currently not served well at all: data management systems, because of their necessary complexity, are expensive and do not perform well on small to medium scale computer systems. As a result, general data management systems are often not used by this user group, and special purpose in-house systems are often developed. A family of database machine products can be developed which provide mid-scale performance and can be trimmed down toward the small user or expanded upwards toward the larger user.

In choosing this market, it is essential to have a good price/performance ratio and be as host independent as possible, since there are many different types of host machines in use by mid-scale users and even more diversity among the small business systems users. The decision to focus the design

toward the mid-scale user dictates various design choices, which are discussed in the following section. The IDM can accommodate moderately large or small users. It is designed to store databases up to 32 Gigabytes. It should achieve transaction rates up to 2000/minute in certain applications but also be capable of being scaled down for users who only need 100 transactions/minute.

## 3. DESIGN TRADE-OFFS

Having chosen to build a database machine emphasizing a low cost/performance ratio, there are a number of fundamental design trade-offs which must be considered. We will explain each one and discuss how the choice of user application influences the decision.

### 3.1. Functionality of the back-end machine

The first decision is what functionality to provide in the database machine. The degree of host independence is closely tied to the level of functionality provided in the back-end machine. The performance improvement derived from using a database machine is related to how much work can be off-loaded to the back-end. Figure 2 shows possible levels of service that a back-end database machine can provide. These are:

- 1) User-programmable, cached controller.  
Allows the user a programming environment in the disk controller where (s)he can control the on-disk processing. Caching can be used to read a track at a time and other such techniques can be used to reduce the apparent access times.
- 2) Search unstructured files for values requested by the front-end.  
This is a simple search engine, and is most suited to applications with data that cannot be structured (such as text processing).
- 3) A record management system.  
This would provide an interface at the record level such as insert record, get next record, etc.
- 4) Provide a basic relational data-management system,

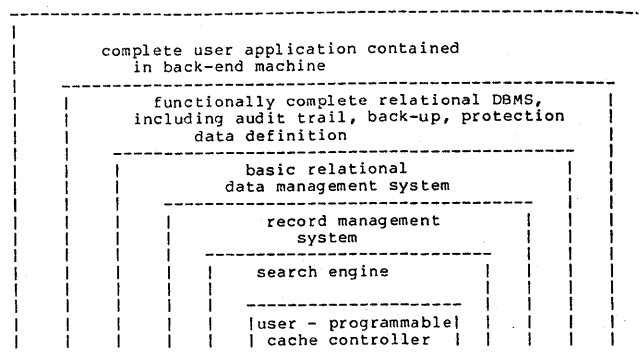


Figure 2—Evolution of back-end intelligence.

with search and update capabilities, interfaced through a high level language.

Host communication time and software support is minimized by such an approach. The database machine can be used most effectively by fully containing a basic database system within it.

- 5) Provide audit trails, back-up and recovery facilities.  
If a data management system is to be contained in the back-end, it should maintain its own back-up and recovery system; otherwise, the host will have communication difficulties knowing what the system is doing, and when.
- 6) Provide protection and data definition facilities.  
Logically, this can be done in the host or in the database machine; doing it in the database machine provides a mechanism that the database machine can use to optimize performance.
- 7) Full end user support.  
If the database machine is to provide a complete data management system, why not make it directly control the terminals and run the application programs, thus eliminating the need for a host?

Figure 2 represents the evolution of increasing back-end intelligence from a disk controller to a back-end machine. Each of the steps makes sense but the crucial issue is which are cost effective and how much do they affect the host system. Until one reaches a level where the back-end is functionally independent from the host, many portions of the host are closely tied to the back-end. Another consideration is that a database machine is a piece of hardware replacing what has traditionally been software. It is not readily subject to user modification. Issues which are closely tied to the end user should be solved in the host since they are likely to require local customization.

A final consideration, and one that cannot be over-emphasized, is that the amount of work which the back-end offloads from the host should be significantly greater than the amount of work needed to communicate with the back-end.

The amount of work done by the back-end in 1, 2, and 3 is not large compared with the overhead of interacting with the back-end. Also, at such a low level, the host processor system is tightly coupled to the low level implementation decisions of the back-end. For these reasons we feel the first significant performance improvement occurs when the host deals with the back-end at a high level (case 4). Case 4 is a database management system with a high level interface. The decision to make the system relational was never questioned.

Providing the additional functionality of 5 and 6 increases the complexity of the database machine but greatly simplifies the tasks done in the host.

The final step, doing everything in the back-end, is counter productive. It casts in concrete parts of the end user interface and furthermore, the back-end processor has no performance advantage for running application programs over any number of very good host processors available today. The database machine's hardware performs extremely well for database

management but offers little to the general programming environment. Running application programs requires a different architecture than running a dedicated database management system.

### 3.2. Storage medium

In order to keep the cost/performance ratio low, providing on line storage in the range of 8 Megabyte to 32 Gigabyte requires the use of moving head disks. They currently are the only non-volatile storage medium with a price/performance suitable for the target market place. The end user costs for standard, moving head disks are \$60 to \$120 per megabyte for large systems (over 100M) and \$100 to \$200 per megabyte for small systems. These costs have been dropping and research has shown that they will continue to drop for some time to come.

### 3.3. Search mechanism

A fundamental part of database management involves searching through objects in the database. The choices for search mechanism can be broken into two categories: complete scanning of the object, or indexing/hashing techniques (which we shall call access methods for the purpose of this discussion). Access methods limit a search based on certain predefined keys. Some commonly used access methods in modern database management systems are B-trees [BAYE70], ISAM [IBM66], and hashing [KNUT73]. In contrast, a complete scan takes any search keys and linearly searches. On traditional systems such a search will take an inordinate amount of time and is impractical except in very small databases. Special database machines, however, can be designed to perform multiple linear searches in parallel. For moving head disks, this requires having multiple heads active at the same time. Each head needs search logic sufficient to search at the transfer rate of the disk. A number of database machines, such as RAP [OZKA78], have been proposed whose speed depends on having many search elements.

The basic trade-off is to compare access methods against multiple search paths, based on expected access patterns and cost. With current technology, the cost of one search mechanism which can perform at disk speeds is substantially less than the cost of multiple search paths employing multiple active heads. Multiple search paths may work for the upper end of the cost/performance curve but they are unaffordable at the other end. Access methods are clearly cheaper; the next step is to determine how they can be expected to perform compared to multiple search paths. Studies have shown that access methods out-perform multiple search path systems on transactions which have a predictable access pattern [HAWT79].

One final trade-off to consider is the impact on the end user. Multiple search paths relieve the Database Administrator from having to determine the physical structure of the database objects, that is, what access methods should be used on what fields. In some cases this is a simple task (for

example employee numbers and employee name will be commonly accessed), but there are other cases when the optimal choice of access methods is extremely difficult to determine. We have opted to leave this burden with the end user in exchange for a system which will be much lower in cost and will have a potentially high performance.

In summary, the use of access methods fits in well with databases which have predictable access patterns. To the degree that this is true, an access method database machine will have a much better cost/performance ratio than a multiple search path machine.

### 3.4. *Low cost processors*

Having decided to use standard moving head disks and access methods, we need processing elements which have comparable speeds. Low cost implies the use of microprocessors but no existing microprocessors have a speed which is satisfactorily matched to the database application. Existing database management systems are frequently execution bound even on fast mainframes. Moving a DBMS onto one of today's 16 bit microprocessors makes economic sense but does not provide reasonable performance for the class of users we have identified.

The trade-off we pursued was to achieve processing rates at least one order magnitude greater than those of microprocessors but at only a modest increase in price. The folklore in processing costs, Grosch's Law [GROS75, CALE79], dictates that this is impossible in the general case. However, if one chooses to do a specific task, not a general one, some surprising results come out. It is possible to structure a DBMS such that an enormous percentage of its time is spent in a small (under 4K) amount of its code. This code is simple in function and specific in nature due to the fact that it deals only with the issues of one task, database management.

By building a 10 MIPS, pipeline machine from standard Shottky TTL logic and microcoding a well defined collection of subroutines, we are able to meet a high performance at a modest increase in price. The special purpose "Database Accelerator" board costs approximately two times the cost of a microprocessor board but in our case has a 30 times increase in performance for the set of code it is intended to perform. Such trade-offs are inherent in special purpose architectures such as database machines.

The high speed of the Database Accelerator enables the IDM to process data as it is coming off the disk. For operations which require disk access, the Database Accelerator will appear to process data with zero real time cost. The operation is limited by the speed of the disk. When only a small transaction rate is required, the Database Accelerator can be removed, reducing the cost of the machine.

### 3.5. *Use of cache memory*

If we want a machine that can run faster than disk speed, it is possible to implement a disk cache using random access memory. This will improve performance only if a disk page

is referenced more than once in a reasonably small period of time. The trade-off is the cost of the cache versus the relative performance improvement. The performance is completely dependent on the amount of "rereferencing." This in turn is highly dependent on the relative ratio of the cache size to the data storage size.

In general most database applications show very little rereferencing. There are, however, certain exceptions [HAWT79]. These include references to the upper levels of index pages, references to system catalog pages (audit trail, locking, data dictionary, etc.) and rereferencing does occur in more complicated user queries. The optimal amount of cache is therefore strongly application dependent. The low end IDM provides a minimum amount of cache (approximately 32K). Additional cache can be included as a user option. Those applications which have a very high transaction rate on a modest size database can incorporate sufficient cache to speed the database machine to near main memory speeds. This ability is consistent with the desire to serve a large range of speed requirements.

### 3.6. *Single/multi-thread*

Both the host independence and the performance of the database machine are affected by the decision of whether to multi- or single-thread the database transactions. A database machine has the choice of either sequentially executing one transaction at a time or accepting multiple transactions and scheduling their execution in a manner similar to what is done in time sharing systems.

To provide an intuitive example, we examine an analogy found in traditional disk controllers. Ignoring the issue of overlapped seeks, a controller takes one operation (read a sector) and does not accept another until the current operation is complete. To enhance performance an operating system will schedule the next disk request from a queue of requests by some strategy which tends to minimize overall seek time. If the selection strategy were moved into the disk controller, it would then appear to be multi-threaded. Disk requests could be made by the host at any time. When an operation was complete, the disk controller would need to tell the host processor not only that an operation was complete, but what operation was completed. The trade-off for doing scheduling in the disk controller is minimal. It would move a small amount of overhead out of the host. It would also allow global scheduling when multiple hosts are connected to one controller.

For a database machine, the trade-offs between single and multi-threading are much more significant. To begin with, how would an operating system decide how to schedule a database transaction? It would require an enormous amount of information about what the transaction does. This amount of information is contrary to the one of the goals of back-end processing, e.g. independence between devices.

The range of times it takes to process a database transaction varies enormously. A typical transaction may require only a few 10's of milliseconds, but other transactions can take minutes or hours.

To allow a mixture of different transactions of differing amounts of work, it is necessary to be multi-threaded and also allow preemption of a transaction. The cost to do this includes adding scheduling code, and room for swapping different transactions in and out of execution. This requires a significant amount of logic in the database machine. Many of the proposed database machines are single-threaded or depend on a cooperating program in the host to coordinate rescheduling.

The decision for the IDM is to provide a multi-threaded environment. This gives a very high degree of independence from the host operating system. Additionally, the problem of where to store transactions which are waiting for resources is solved by "stealing" memory from the cache. At any point in time a percentage of the cache memory is dedicated to transactions and to caching disk activity. The percentage is allowed to change dynamically. For example, at a particular point in time there may be only one transaction running and it may have nearly all the cache memory to use for processing. Similarly, if there are many, short transactions, the cache will be allocated mostly for transaction swapping. A heuristic algorithm is used to determine how best to utilize the cache resource. This fine degree of control is best done in the database machine. The host operating system has very little to do except pass program requests to the IDM and pass results back to the program. In DMA interfaces, this overhead is trivial, making the driver program in the host operating system very simple.

#### 4. CONCLUSIONS

In this paper we have examined a number of the fundamental design decisions in designing database machines. We began by identifying the class of applications which the database machine was to address, namely, the mid-range user who has applications with predictable access patterns. To build a machine with a good cost/performance range over a wide range of storage requirements requires moving head

disks and access method search techniques. To achieve high execution speed, special purpose hardware (the Database Accelerator) is used. There are specific applications where a variable size cache can greatly improve performance. Finally we have illustrated why a database machine must be multi-threaded and the rationale behind making the database machine support a functionally complete, relational database management system.

#### BIBLIOGRAPHY

- [BAUM76] Baum, Richard I., Hsiao, David K., and Kannan, Krishnatmuti, "The Architecture of a Database Computer - Part I: Concepts and Capabilities," Technical Report OSU-CISRC-TR-76-1, Computer and Information Research Center, The Ohio State University, Columbus, Ohio (National Technical Information Service Number AD-A034 154).
- [BAYE70] Bayer, R. and McCreight, E., "Organization and Maintenance of Large Ordered Indices," *Proc. 1970 ACM-SIGFIDET Workshop on Data Description, Access, and Control*, Houston, Texas, Nov. 1970.
- [CALE79] Cale, E. G., et al., "Price/Performance Patterns of U.S. Computer Systems," *CACM*, Volume 22, Number 4, April 1979.
- [DEWI79] Dewitt, D. J., "Query Execution in DIRECT," *Proceedings, SIGMOD International Conference on the Management of Data*, 1979, pp. 13-22.
- [GROS75] Grosch, H. A., "Grosch's law revisited," *Computerworld* 8, 16, April 16, 1975.
- [HAWT79] Hawthorn, Paula, "Evaluation and Enhancement of the Performance of Relational Database Management Systems," Electronics Research Laboratory, University of California at Berkeley, Berkeley, Ca., Memo Number M79-70, December, 1979.
- [IBM66] IBM Corporation, "OS ISAM Logic," IBM, White Plains, N.Y., 1966, GY28-6618.
- [KNUT73] Knuth, D. *The Art of Computer Programming*, Vol. 3, Addison-Wesley, Reading, Mass., 1973.
- [LIPO78] Lipovski, G. J., "Architectural Features of CASSM: a Context Addressed Segment Sequential Memory," *Proceedings, Fifth Annual IEEE Symposium on Computer Architecture*, April, 1978.
- [OZKA78] Ozarahan, E. A., Schuster, S. A., and Sevcik, K. C., "Performance Evaluation of a Relational Associative Processor," *AFIPS Conference Proceedings*, Vol. 44, 1975, pp. 379-388.
- [STON79] Stonebraker, Michael, "MUFFIN: A Distributed Database Machine," Electronics Research Laboratory, University of California at Berkeley, Berkeley, Ca., Memo Number M79-28, May 1, 1979.



# DIALOG—A distributed processor organization for database machine

by BENJAMIN W. WAH and S. BING YAO

*Purdue University*  
West Lafayette, Indiana

## 1. INTRODUCTION

The conventional physical storage mechanism of a computer system is usually comprised of a memory hierarchy that stores program and data. The requirement for high performance and low cost is achieved through a combination of memories of different speeds. By automatically managing the files so that the most frequently used files reside in fast storage, an overall speed comparable to the speed of the fastest memory can be achieved. However, with the applications of large databases, the maintenance of large files on a conventional memory hierarchy becomes increasingly difficult. Most database applications perform a small number of simple operations on a large amount of data. Usually only a small fraction of the data accessed is required by the application. It is more cost effective to perform database operations directly on the data in the secondary storage in order to avoid the transfer of unnecessary data across different levels of the memory hierarchy. The Database Machine (DBM) is the result of an architectural approach which distributes processing power closer to the devices on which data are stored and offloads database processing functions from the main computer [LAN79].

The feasibility of database machines has greatly improved with recent advances in the computer system device technology. The number of components per chip is approximately doubling each year and the CPU speed is growing exponentially. At the same time, the cost per unit of memory is decreasing and a wide variety of new storage devices, which include CCD memory, bubble memory, and the electron beam access memory, are becoming available. It is now possible to design inexpensive processing elements to be embedded within these new storage devices. Disk technologies have also been improved and it is possible to provide inexpensive secondary and archival storage to the computer system. On the other hand, the software costs and overheads are usually a deterrent factor for computer system development. This cost is becoming dominant in present data processing systems, and is expected to increase rapidly to an even higher percentage in the near future. The replacement of expensive database system software with custom built hardware is therefore a feasible and desirable trend in the future.

The database machine proposed so far can be divided into three types. The first type is the backend system which utilizes mini-computers to enhance the database processing of large host computers. The functions of the backend system include access validation, storage management, concurrency control and I/O control. However, the principle of the design is basically the same as a conventional database system. Large amounts of data not required by the database application are accessed and transferred to the mini-computers. The use of a backend machine is only a temporary method to extend the processing power of a large CPU.

The second type of database machine utilizes the single instruction multiple data stream (SIMD) principle. This concept is extended from backend machines in which the database processing functions are moved to a lower level. The characteristics of this design are that simpler, less costly processors that are dedicated to a small block of data are used. This concept, when applied to the storage cells (modules), is exemplified by the Logic-Per-Track device in which processing logic are duplicated for each track and the keys in different tracks are searched in parallel. Examples of this design are TapeDRUM [HOL56], Slotnick's Logic per Track Disk [SLO56], RAPID [PAR72], CASSM [LIP78, SU79], RAP [OZK77, SCH79], RARES [LIN76], DBC [BAU76, KER79, BAN79], and Chang's Major/Minor Loop Machine [CHA78]. In this type of design, the database workload must be heavy in order to keep the parallel resources fully utilized. For large databases, the amount of replicated processor elements may be prohibitively large. The degree of parallelism in this type of design is also limited by the number of read/write heads, i.e., the number of data streams that can be read in parallel. This concept results in an expensive memory device with parallel read/write heads. Moreover, the processors under most of these designs are quite general with limited amount of communications. Lastly, if the database machine is built on a disk, the processors must be extremely fast because many fast signal translations are needed in order to perform real time processing and disk marking.

If the replication goes further down to the bit level, an associative memory results. Associative memories are usually very expensive and, therefore, can be only shared by swapping in data when needed. This is exemplified by RELACS [BER79] in which STARAN is used as the associative



memory. However, this design experiences the usual problem of data swapping in a memory hierarchy, and there is no provision to select data before it is accessed and transferred to the associative memory. The throughput of the system is therefore quite limited. Further, I/O lines must be extensive in order to load the associative memory in parallel.

The last type of database machine design is based on the Multiple-Instruction-Multiple-Data System (MIMD) principle. In this architecture, which is exemplified by DIRECT [DEW79], the processing logics are separated from the storage modules and are interconnected with an array of CCD memories through a cross bar switch. This design offers more flexibility and better load balancing and allows the processors to be shared among the storage modules. Because of the fact that each processor can access multiple storage modules simultaneously, it is easier to perform database operations which require multiple files to be coupled, e.g. a multi-relation join. Further, expansion is easy and modular. However, this design suffers from the same disadvantage as the associative memory when the size of the CCD memory is not large enough, in which case excessive swapping will occur. Again, since the processing logic is removed from the storage device, large amounts of unnecessary data are accessed and transferred.

We note that the previous designs are built around a single type of storage device, e.g. disk, CCD, etc. Intramodule operations can be performed very efficiently because they do not utilize the I/O bus. However, inter-module operations often result in a bottleneck at the I/O bus. At any one instant, only one inter-module operation can be processed because the designs are essentially SIMD. In some designs, e.g. DIRECT, where the problem of I/O bus has been solved by using a simple cross bar switch, the operations are expensive because data have to be transferred from the mass storage to the CCD storage modules. Only when a sizable amount of operations are performed on the file transferred would the transfer be cost effective.

In this paper, we propose a design of a backend database machine, DIALOG(DIstributed Associative LOGic database machine) which addresses some of the problems mentioned above. We want to design intelligent but simple processing logic so that they can be replicated on the storage modules. Algorithms such as select, project and join will be implemented in hardware so that they can be processed very efficiently. These processors work directly on the storage devices, so that the amount of data transfer is kept at a minimum. A network is proposed which provides a uniform medium to connect heterogeneous memory devices together.

The DIALOG database machine is designed with the following design goals in mind. First, the system should be extendable and be able to support very large on-line databases in the future. Second, the design must have high performance and the cost should be low by replicating a few simple devices. Third, the system should use existing memory technologies in the design such that it can be implemented now, and the design should be able to evolve as new memory technologies are available in the future. Fourth, the system should accommodate heterogeneous storage devices

such that files with different workload and sizes can be stored in the devices with appropriate speeds and sizes. Fifth, the system should implement low level operations (such as select and join) and facilitate higher level query optimization. Finally the system should be able to interface with the host computer in multiple data models (such as relational, hierarchical, and network data models [COM76]).

This paper is divided into five sections. Section 2 provides the overall architecture of the system. It also illustrates how an inter- or intra-module operation is performed. Section 3 describes the architecture of the data module. The functions to be performed in a module are partitioned into the select processor, the associative processor, the join processor and the communication processor. Section 4 presents an analysis on the buffer size required. Section 5 provides some discussions of this system and compares this system against other systems. Lastly, Section 6 gives some concluding remarks.

## 2. SYSTEM ARCHITECTURE

The secondary storage, where the bulk of the database is stored, is usually made up of multiple types of storage devices like disks and tape drives. As the storage technology advances, devices such as CCD memories, bubble memories, EBAM's, etc. could become part of the storage system. Each of these storage types has different capacities and speeds. Since many database operations require data stored in multiple files, it is often necessary to access these files simultaneously. One approach is to transfer all the required data for an operation to a uniform storage device such as CCD memories, which are connected to a set of processors, before processing it. However, there is a significant overhead associated with this data transfer, especially when the original storage device has no selection capability. Further, if the intermediate device is not large enough to accommodate the entire file, costly multiple passes have to be made.

An alternative approach is to send the required data directly from the storage device on which the file is stored to a second storage device on which the file is to be processed. The time to stage these files to an intermediate storage device can be saved. If the storage devices are enhanced by sufficiently powerful processing elements, the results of database operations can be output at a very high speed. In order to facilitate the transfer of data, a communication network must be designed to connect the heterogeneous devices together.

In this section, we describe the general architecture for the latter approach to database machine design. The design of the network will be examined and the processing capability of the machine will be discussed.

### A. General architecture

The general architecture of the system is shown in Figure 1. Data in the system is stored in the *data modules*. Each

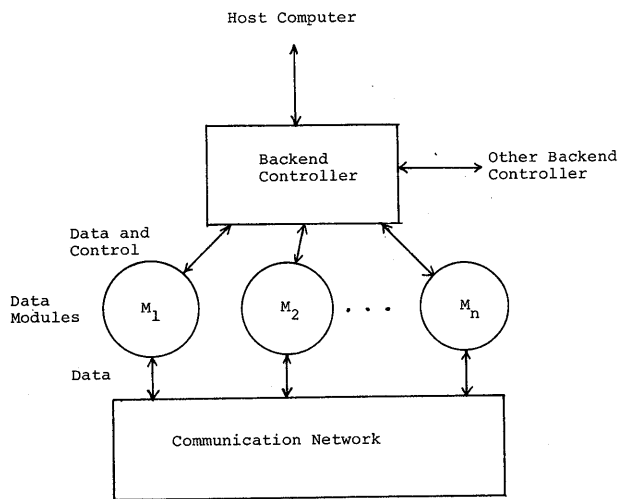


Figure 1—System architecture of a cluster in DIALOG.

data module is consisted of a storage device and an associative processor. Data modules are connected together by an interconnection network. Only data can flow across this network. The cluster of data modules are connected directly with a backend controller which allows both data and control communication. The backend controller provides communication between the cluster of data modules and the host computer. Queries expressed in a high level representation are sent from the host to the backend controller and query responses are returned to the host.

The major functions of the backend controller of a cluster include pre-processing and optimizing the queries, looking up system directory, establishing links between data modules, initiating and scheduling operations within each data module, receiving and buffering output from data modules, managing the sharing of resources, and initiating rollback and recovery as system components fail. All of the above functions will be implemented in software.

Depending on the number of data modules in the database machine, the interconnection network may become quite complex. It may be necessary to group data modules into multiple clusters. The backend controllers of these clusters are connected using a network of the same design as that developed for interconnecting the data modules within a cluster. A higher level backend controller is used to coordinate the controllers of all the clusters.

An example of this hierarchical network is shown in Figure 2. We note that the communication between data modules of different clusters is achieved by sending data through the backend controllers and a high level communication link. The highest level backend controller assumes the responsibility of communicating with the host computer. The operation of the network in each level of the hierarchy behaves in the same fashion as the network in the lowest level which will be discussed later.

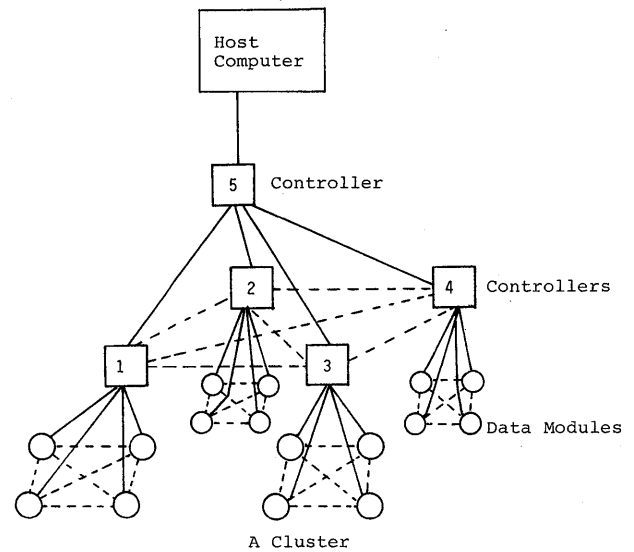


Figure 2—An example of the hierarchical network in DIALOG.

### B. Processing capabilities

Although DIALOG is designed to support multiple data models, only the database operations defined on the relational data model [COD70] are discussed in the present paper. The relational operations currently included in the design are: select, project, join, union, and cartesian product.

Operations that require only one file, such as selecting records that satisfy a given predicate and projecting on certain attributes are usually resolved within one data module. Database operations that require the cross-referencing of files stored in two data modules (such as join and cartesian product) are performed by sending all the required records to one of the data modules and then processed there. In the case when the two files to be processed reside in the same data module, one of the files can be retrieved into the input buffer. The processing continues as if the file in the input buffer were received from a different module.

Database operations that access multiple files in several data modules are decomposed into a set of two-module operations which can be processed either in sequence or in parallel. The decomposition and scheduling of database operations and the routing of files are determined by the backend controller.

### C. Network design

The design of the interconnection network has an important implication on the throughput of the system. A ring network is the simplest form of communication network, but it could cause a lot of contention. If the conflict is resolved by a centralized controller, the controller may become a bottleneck of the system. If the conflict is resolved in a dis-

tributed fashion, the software for the communication protocol may be complicated. It is therefore desirable to have a network that is conflict free and requires a small amount of software development.

The simplest conflict-free network is the fully connected network, although this network may seem very expensive because the number of links grows as  $n^2$  where  $n$  is the number of data modules. However, these links are merely serial lines governed by a central clock, and their costs are minimal. An implementation of the link between two data modules is shown in Figure 3. The link can be established by the controller which sends a command to the communication processing modules of the two data modules. The communication processors at the two data modules then set the multiplexor and the demultiplexor and the link is established. The links can only be broken by a command from the controller. The communication processor is responsible for managing the buffer pool, detecting and correcting errors in the data, informing the controller if a non-recoverable error is detected, and signaling the source of a data transfer to stop when the buffer is full. Since the communication between two data modules is governed by a central clock, any type of memory device can be connected to the network. The design can also be modified to broadcast data to several modules simultaneously.

With this network design, global data transfer is very easy. Referring to Figure 2, suppose one of the data modules in cluster 1 wants to send data to a data module in cluster 3. Since there is no direct path of communication between these two data modules, data from cluster 1 will be sent first to its controller. Since there is a direct path of communication between controllers of clusters 1 and 3, this path is established by controller 5, the controller in one level higher than controllers 1 and 3. The file transfer can be carried out without the intervention of controller 5. Of course, this approach can result in excessive overhead for the lower level controllers when inter-cluster transfers are large. However, by carefully allocating the relations to the clusters so that the inter-cluster communications are minimal, this technique is still a cost effective approach.

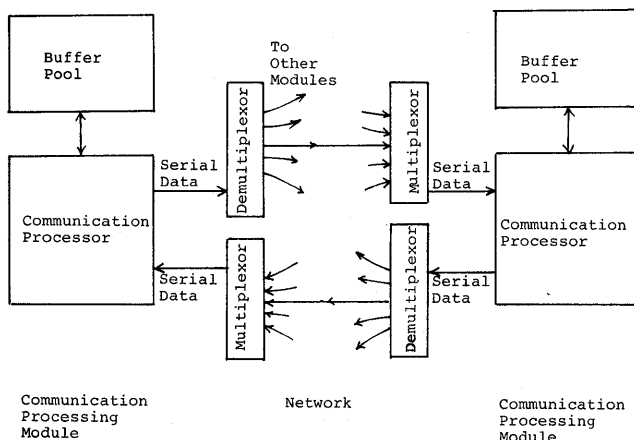


Figure 3—A communication link between two data modules.

### 3. DATA MODULE DESIGN

In this section, we describe the internal design of the data module. The function of the data module is to store the data and to process relational queries directed on the stored data. The design should allow both inter- and intramodule queries to be performed efficiently. Furthermore, the design should be modular and functional so that current microprocessor and VLSI technologies can be used in the design.

#### A. General data module architecture

The general architecture of a module is shown in Figure 4. There are five basic sub-modules of the system, namely: the physical storage device which contains the database; the selection processing module which processes projection and selection operations; the associative processing module which compares the output from selection processing with the stored search keys and passes successful matches to the join processing module which then produces the join; and lastly, the communication processing module which manages the buffer pool and communicates with other data modules in the network. We describe each of these sub-modules here.

#### B. Data allocation in the physical storage device

The physical storage device is made up of multiple circulating loops of data with a single read/write head for each loop. The loops may be simple which can model devices like CCD memories. It may be more complicated such as the major/minor loop of a bubble memory, the LARAM organization of the CCD memory, or a cylinder of a disk. The access time distribution for accessing a piece of data within the loop consists of two components, the time to switch to read/write a particular loop and the time to shift the data in the loop to the read/write mechanism. Different memory devices have different time delays for each component. In the case of a single loop, data can only be read out serially. However, in the case of a multi-loop organization, data may be read out one loop at a time with electronic switching between the loops. This is characterized by a cylinder of a disk or a LARAM organization of CCD memory. Data may also be read out serially from all the loops simultaneously. This type of memory is characterized by a multi-chip bubble or CCD memory and the Parallel-Transfer-Disk [AMP78]. However, in some implementations, it may not be possible to synchronize all the parallel outputs on the device (e.g. a cell may be bad in one of the loops and unless all the corresponding cells in other loops are marked bad, the outputs will be out of synchronization). It is therefore not practical to organize the database such that a tuple of a relation is read out bit parallel but word serially. The organization of the database in a multi-loop device is therefore chosen to be bit-serial and word-parallel, that is, multiple tuples will be available simultaneously at the read heads. In this case, since we assume that the selection processing module can

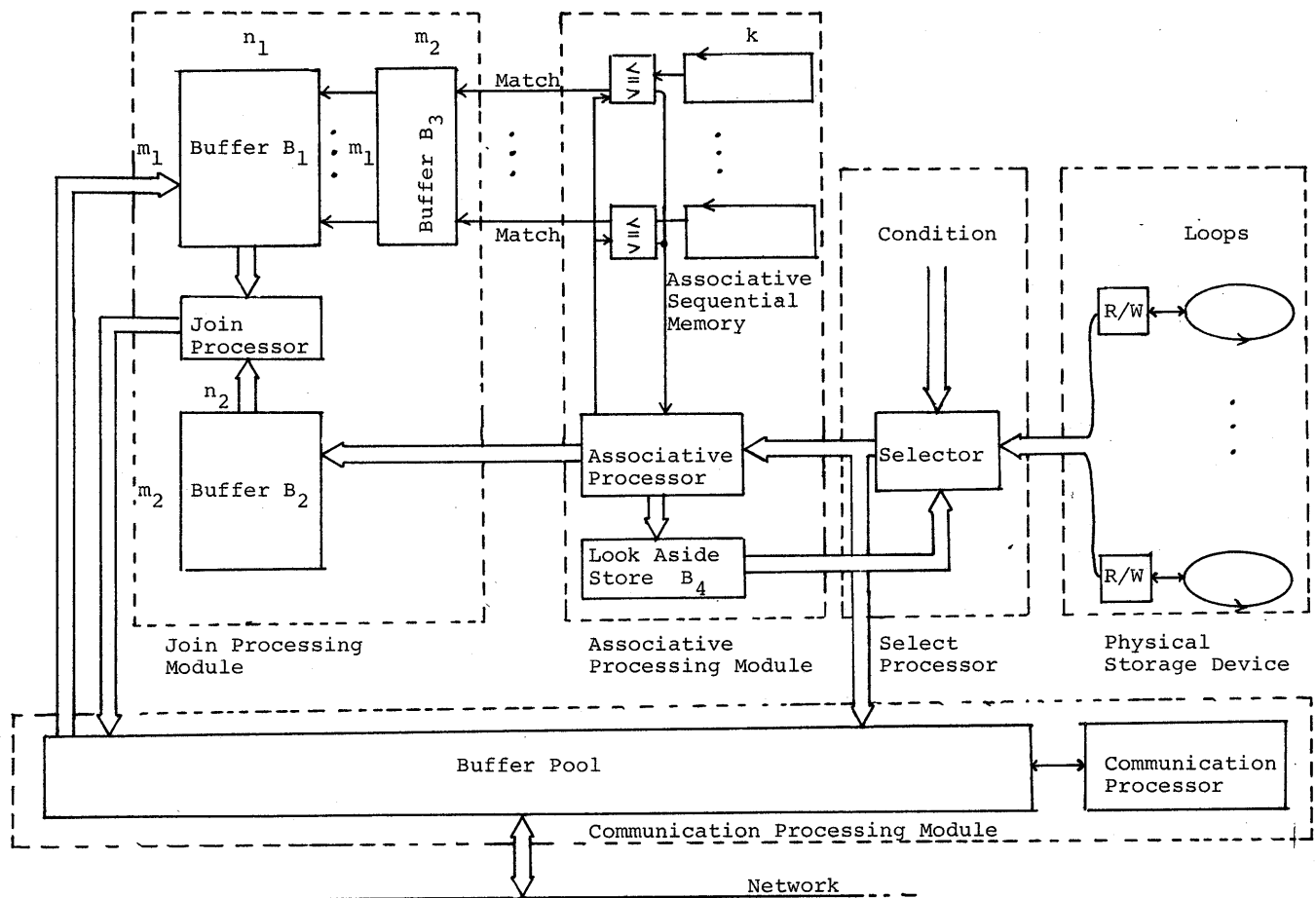


Figure 4—Architecture of a data module.

process one tuple at a time, a buffer is added between the physical storage device and the selection processing module so that all the parallel tuples are sequenced sequentially. Of course, the speeds of the other processing modules have to be increased from a single loop model correspondingly. With the use of a buffer, a multi-loop device behaves like a single loop device. We conclude that a single loop model is sufficient for our design.

### C. Selection processing

In selection processing, tuples are examined in the database one at a time and only those tuples satisfying certain predicate will be output. This output can be sent to the buffer pool or it can be sent to the associative processing module. When the output is sent to the buffer pool, it may be sent to another data module in the system for join processing or it may be used to join with another relation on the same data module. Projection is an operation similar to selection. However, we assume that duplicates will not be removed in the data module because of the limited buffer capability.

### D. Join processing

In performing a join on this system, both the associative processing module and the join processing module will be used. The buffer data allocations are as follows (see Figure 4). Buffer  $B_1$  contains  $m_1$  tuples, each of  $n_1$  bytes. The contents of  $B_1$  is filled from the buffer pool and they represent part of the tuples of a relation coming from the same or a different data module in the system. These tuples are used to join with the tuples coming from the output of the selection processing module. The size  $n_1$  is chosen so that it can accommodate the largest tuple used. The size of  $m_1$  will be determined in Section 4.

The associative sequential memory in the associative processing module is made up of  $m_1$  circulating loops of size  $k$ . It contains the keys of the join domain of the corresponding tuples in buffer  $B_1$ . It is a one to one correspondence and the keys of the  $i$ th tuple in  $B_1$  is contained in the  $i$ th loop of the associative sequential memory. The size  $k$  is chosen so that the loop can accommodate the largest key used. For smaller keys, multiple copies may be made in the same loop to allow faster access.

The associative processing module compares the serial output tuple from the selection processing module associatively against all the keys in the associative sequential memory. If at least one match occurs, it indicates that a join is possible between this tuple and the corresponding tuple in  $B_1$ . This tuple is put in buffer  $B_2$  and it will be joined with the corresponding tuples in  $B_1$  when all the previous joins are performed.  $B_2$  contains a queue of  $m_2$  tuples, each of  $n_2$  bytes that can be used to join with keys in  $B_1$ . The size of  $n_2$  is chosen so that it can accommodate the largest tuple used ( $n_1$  and  $n_2$  are probably chosen to be identical). The size of  $m_2$  is determined in Section 4.

As the match is done in the associative processor and a successfully matched tuple is added to  $B_2$ , a word is written into buffer  $B_3$ .  $B_3$  is an  $m_1$  by  $m_2$  bit matrix. A column in  $B_3$  represents the set of tuples in  $B_1$  which match with the corresponding tuples in  $B_2$ , where a 1 indicates a match and a 0 indicates a no-match.

The example in Figure 5 illustrates the functions of  $B_1$ ,  $B_2$ ,  $B_3$  and the associative sequential memory where  $m_1$ ,  $m_2$  are chosen to be 2 and  $n_1$  and  $n_2$  are chosen to be 12. In this example, suppose the following join is to be performed.

RETRIEVE (A.city, B.pname): A.p# = B.p#

$B_1$  contains part of the tuples of relation  $A$ , and  $B_2$  contains part of the tuples of relation  $B$  which are to be joined with the tuples in  $B_1$ . For the left column in  $B_3$ , it indicates that the first tuple in  $B_2$  can be joined with the first tuple in  $B_1$ . Similarly, the right column in  $B_3$  indicates that the second tuple in  $B_2$  can be joined with the first tuple in  $B_1$ . The loops in the associative sequential memory contain the keys of the corresponding tuples in  $B_1$ . Note that in our example, the  $p\#$  domains in  $B_1$  and  $B_2$  are never output from the join processor and they are not used by the join processor to determine which of the tuples in  $B_1$  and  $B_2$  are to be joined (because  $B_3$  provides this information). These two domains therefore do not have to be included; however, they are shown here for illustration.

The join processor performs the join between the tuples in  $B_1$  and  $B_2$  by picking up the first tuple in  $B_2$  and joining it with all the tuples with matched keys in  $B_1$ , before proceeding to the next tuple in  $B_2$ . The results of the join processor are routed to the buffer pool.

E. Associative sequential memory

The design of this associative sequential memory is based on the design by Ramamoorthy, Turner and Wah [RAM78,

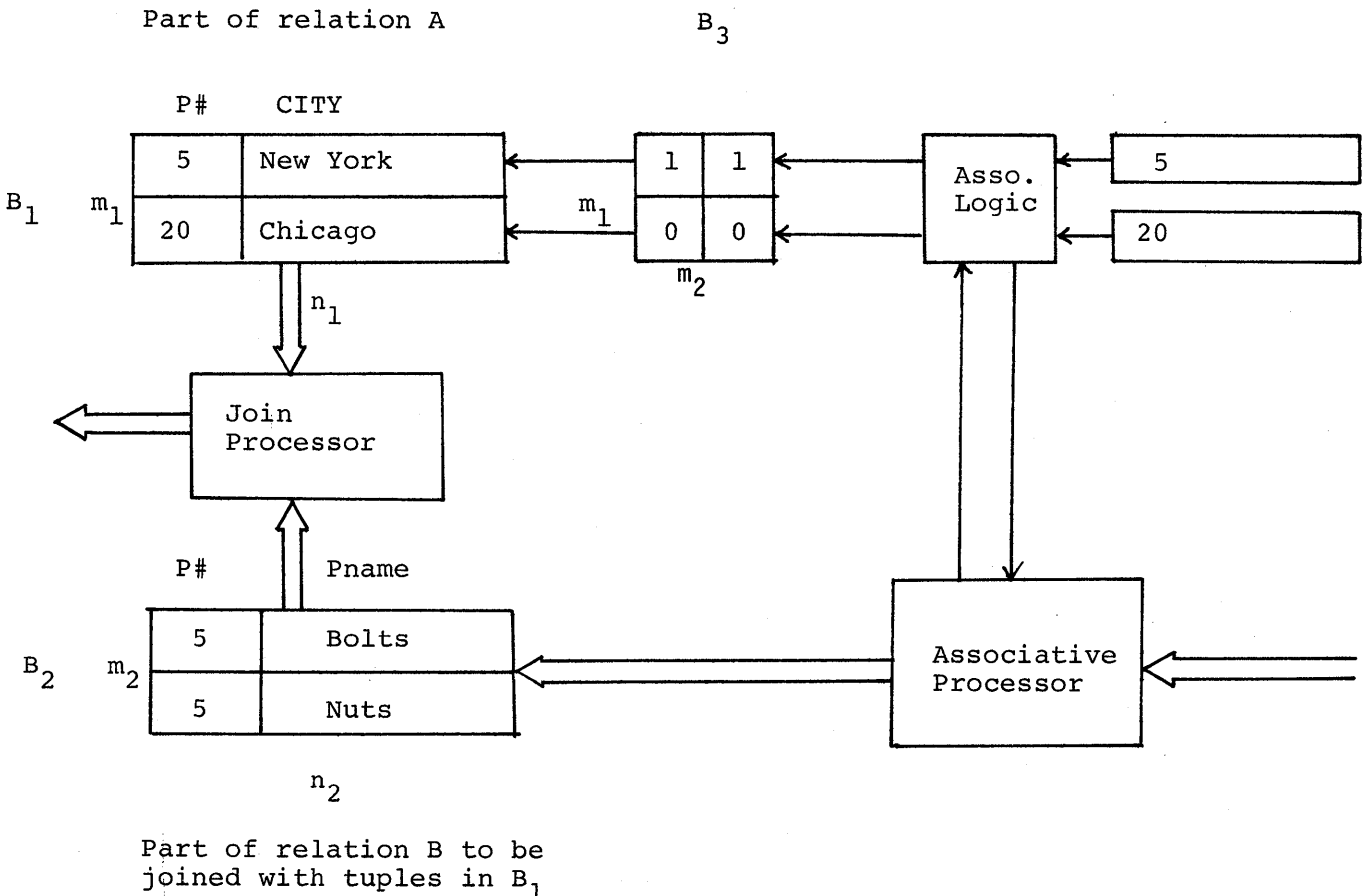


Figure 5—Example to illustrate the function of buffers  $B_1$ ,  $B_2$ ,  $B_3$  and the Associative Sequential Memory.

WAH79]. The sequential memory is made up of multiple loops of circulating bits shifting in synchronism that may be CCD memory or bubble memory. In this design,  $m_1$  keys are stored in the memory, with one key occupying each loop. Associative logic capable of performing equality, threshold and proximity searches are put at the read/write head of each loop. The architecture of this design is shown in Figure 6. The detailed cell logic is shown in Figure 7. During a clock period, a bit-slice of these  $m_1$  keys is shifted out from the sequential memory and is compared associatively with the output from the selection processing module. The enable signals are stored in temporary flip flops. As the bit-slices are shifted out, most significant bit first, the bit-slice, together with the stored enable signals, generate a new set of enable signals which are stored back into the temporary flip flops. There are three significant improvements in this design as compared with conventional logic-per-track device. First, the additional logic for each loop is very small and therefore the cost increase is minimal. Further, the cell logic is simple enough to be implemented on the same chip as the memory elements with only a minimal cost increase. Second, there are no communication lines between two adjacent loops and therefore the number of loops can be modularly expanded. Lastly, the number of loops is not governed by the number of read/write heads on the physical storage device. Previous database machine designs assume that the number of associative logic is replicated for each head of a disk which therefore limits the maximum degrees of parallelism. In our design, the number of heads on the disk is not a factor in the number of loops used in the associative sequential memory. The degree of parallelism can therefore be improved significantly.

One problem that exists with this design is to choose the loop size of the associative sequential memory so that the

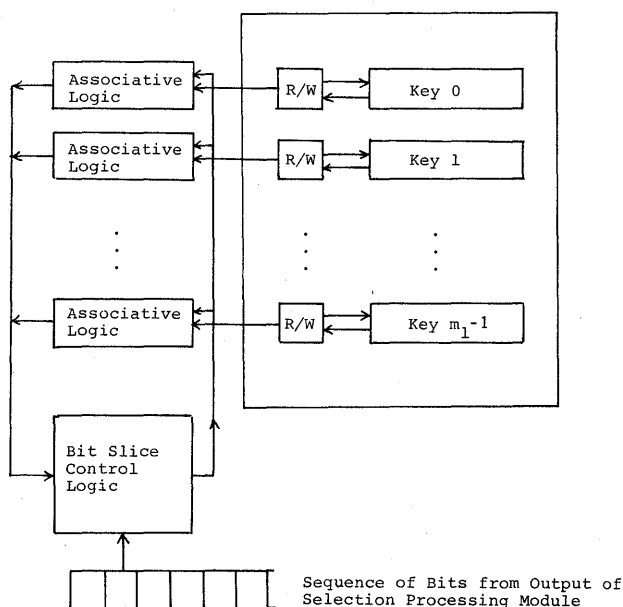


Figure 6—Associative Sequential Memory [RAM78, WAH79].

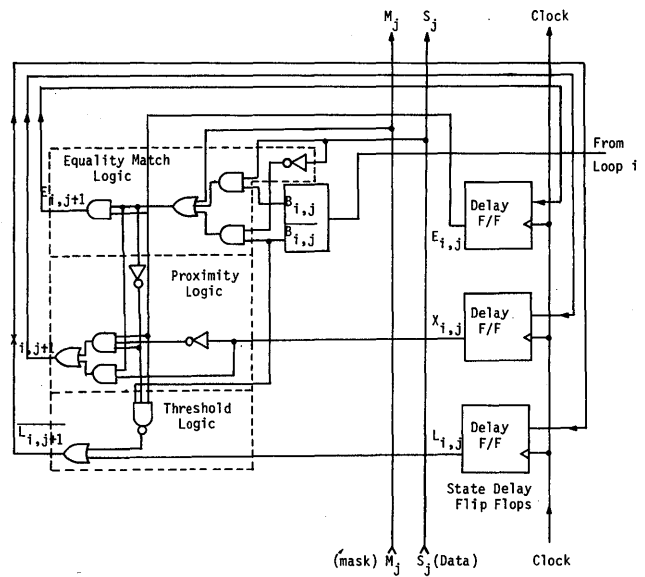


Figure 7—Associative logic for Associative Sequential Memory with equality, greater than, less than and proximity searches [RAM78, WAH79].

largest key can fit into the loop. However, if the loop size is too large and the time it takes for the loop to make one complete revolution is longer than the time it takes for a tuple to be shifted out, the associative sequential memory may not be able to catch up with the output rate of the selection processing module. In this case, one simple solution is to duplicate the keys in the loop so that the time to reach the key is always shorter than the time to shift out a tuple. Buffers may also be placed between the associative processing module and the selection processing module to smooth out the irregularities. Since the output of the associative processing module goes to the join processing module and there is a finite probability that  $B_2$  is full, the associative processor may not be able to continue to process the tuples once  $B_2$  is full. In this case, a lookaside buffer,  $B_4$ , is used to store the missed tuples. This buffer is a queue with each element being a tuple identifier. The size of  $B_4$  is chosen so that it would accommodate all the missed tuple identifiers.

F. Join processor

The join processor uses buffers  $B_1$ ,  $B_2$ , and  $B_3$  to produce the join outputs. One characteristic about  $B_2$  and  $B_3$  is that they are variable size queues. This feature may be implemented by a hardware or a software linked list. Further, accesses to each column of  $B_3$  must be made in parallel. The join processor uses one column of  $B_3$  to find out which tuples in  $B_1$  are to be joined with a tuple in  $B_2$ . This is a conventional multiple match resolution problem. We assume that a sequential search is made to find all the 1's in a column of  $B_3$ . However, tree circuits [FOS68] or associative memory [RAM78, WAH79] can be used to do the multiple match resolution.

We have discussed in this section the detailed design of

the data module. Finally, when the tuples in  $B_1$  have been matched with all the tuples in the physical storage device (e.g. a cylinder, a loop, etc.), the contents of  $B_1$  and the associative sequential memory are switched to a new set. As data are moved into the buffers, they are also moved in parallel into buffers  $B_1$  and the associative sequential memory. The processing of a join is therefore pipelined and the throughput of the system can be greatly enhanced. The throughput of the system is a function of the sizes of  $B_1, B_2, B_3$ , the tuple size and the bit rate (the rate at which tuples are selected out from the physical storage device). We present in the next section an analysis for the sizes of  $B_1$  and  $B_2$  if the join outputs are to be made in disk transfer rate.

4. APPROXIMATE ANALYSIS OF BUFFER SIZE

In this section, we perform an analysis on the size of the buffers. The sizes of  $B_1, B_2$  and  $B_3$  are critical factors in our design. We first establish an upper bound on the average queue length in  $B_2$  and  $B_3$  given the size of  $B_1$  and the probability that a match occurs between the contents of  $B_1$  and a tuple output from the selection processing module. We further assume that  $B_2$  and  $B_3$  are very large. Using the upper bound on the queue size of  $B_1$ , we can establish a size for  $B_2$  and  $B_3$  so that, under most circumstances,  $B_2$  and  $B_3$  will not be full. The approximations are then compared with the simulation results.

Let:

- $P$  = probability that a tuple of relation  $A$  in  $B_1$  will be joined with a tuple of relation  $B$ ;
- $\lambda_a$  = length of each tuple of relation  $A$  in  $B_1$ ;
- $\lambda_b$  = length of each tuple of relation  $B$  in  $B_2$ ;
- $\lambda_a'$  = length of each tuple of relation  $A$  after restriction;
- $r$  = transfer rate of physical storage device;
- $r_0$  = desired join output rate;
- $s_1$  = number of tuples in  $B_1$  (= size of buffer  $B_1, m_1$ );
- $s_2$  = number of tuples in  $B_2$  (size of buffer  $B_2$  is assumed to be very large);
- $s_2 = E(S_2)$ .

When the join processor picks up a tuple in  $B_2$ , there is at least one match with a tuple in  $B_1$  (otherwise, the associative processor would not have passed this tuple to  $B_2$ ). The time to produce all the corresponding joins depends on the number of matches in  $B_1$ . For  $i$  equal matches ( $i=1,2,\dots,s_1$ ) with a probability of  $\binom{s_1}{i} p^i (1-p)^{s_1-i} / (1-(1-p)^{s_1})$ , the time to produce all the joins are  $(\lambda_a' + \lambda_b) * i / r_0$ . On the other hand, the inter-arrival time distribution is geometrically distributed, with a probability of  $(1-p)^{s_1(i-1)} * (1-(1-p)^{s_1})$  for an interarrival time of  $(i * \lambda_a) / r$  ( $i=1,2,\dots$ ). This system is in effect a GI/G/1 queue. The queue is shown in Figure 8. We use Marshall's inequality [MAR68] to find the average delay time that a tuple spends in  $B_2, d$ .

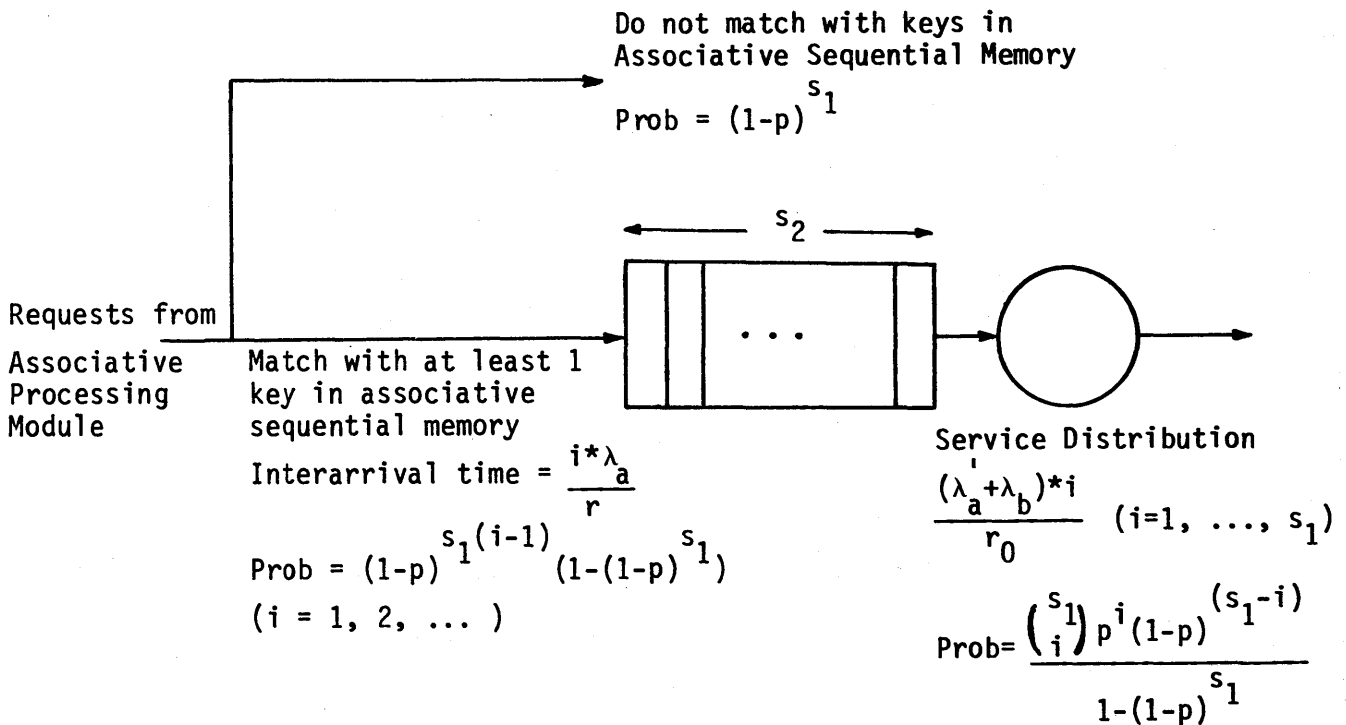


Figure 8—Queue to model operation of associative processing module and join processing module.

$$d \leq \frac{\lambda(\sigma_a^2 + \sigma_g^2)}{2(1-\rho)} \quad (\text{Marshall's Inequality})$$

where  $1/\lambda$  and  $\sigma_a^2$  are the mean and the variance of the inter-arrival time,  $1/\mu$  and  $\sigma_g^2$  are the mean and the variance of the service time and  $\rho = \lambda/\mu$ . After evaluating  $d$ , we can further apply Little's Formula [LIT61] to find a bound on  $\bar{s}_2 (\bar{s}_2 = \lambda d)$

$$\bar{s}_2 \leq \frac{\left[ \bar{p} + \left( \frac{\lambda_a' + \lambda_b}{\lambda_a} \right)^2 \left( \frac{r}{r_0} \right)^2 s_1 p (1-\bar{p}) \left[ p(s_1-1) + 1 - \frac{s_1 p}{(1-\bar{p})} \right] \right]}{2 \left[ 1 - \frac{r}{r_0} \left( \frac{\lambda_a' + \lambda_b}{\lambda_a} \right) s_1 p \right]}$$

where  $\bar{p} = (1-p)^{s_1}$ . Assuming that  $\lambda_a = \lambda_b = \lambda_a'$ ,  $r = r_0$ , we have,

$$s_2 \leq \frac{\bar{p} + 4s_1 p (1-\bar{p}) \left[ p(s_1-1) + 1 - \frac{s_1 p}{(1-\bar{p})} \right]}{2[1-2s_1 p]}$$

Simulations are also carried out on the queue in Figure 8. The simulation program is written in ASPOL and is run with 2000 requests for each pair of  $p$  and  $s_1$ .

Table I shows the difference between the simulation results and the estimated bounds for a sample of values of  $s_2$  for various values of  $s_1$  and  $p$ . It is seen that the estimations, although not accurate, produce a close enough bound for  $s_2$ . We have also indicated in Table I a column that indicates the value of  $s_2$  such that  $Pr(\text{queue size} < s_2) \leq 0.95$ . These

values can be used to determine a fixed bound on  $s_2$ . For example, with  $p = 0.01$ , which means that there is a chance of 1 percent for a particular tuple in relation  $B$  to be joined with a tuple in relation  $A$  in  $B_1$ ,  $s_1$  and  $s_2$  can be chosen to be 50 and 36 respectively. For these values of  $s_1$  and  $s_2$ , the utilization of the server (join processor) is 0.994, which means that the join output is indeed coming out at the disk transfer rate. However, the choice of  $s_1$  and  $s_2$  is not arbitrary. If  $s_1$  is set to be too large, the time to switch the contents of  $B_1$  is large which induces large overheads on the system. On the other hand, if  $s_1$  is too small, then there may not be enough requests in  $B_2$  and therefore the join processor may be idle most of the time. It is therefore necessary to choose  $s_1$  as small as possible while giving a reasonably high utilization for the join processor. The choice is driven by the value of  $p$  which is dependent on the number of tuples in the relations and the number of distinct join keys in the two relations. For large relations, this is usually small. Table I shows some simulation results with  $p$  set to 0.001. It is seen that with  $s_1$  set to 160, the utilization of the server (join processor) is 0.317 and  $s_2$  can be set at 1. This means that the join processor will be idle 69 percent of the time and the output rate is 31 percent of the disk transfer rate. With  $s_1$  set to be so large, there is a tremendous switching overhead in the system. In order to remedy this, multiple sets of  $B_1$  and associative sequential memories can be implemented in each data module. The join processing operations on one set of  $B_1$  can be overlapped with the loading of another set of  $B_1$ . The switching overhead on  $B_1$  is therefore overlapped with the join processing. This is feasible because the communication processor in each data module can be designed to send and receive from different data modules so that as the join processor is producing outputs to be sent to one data module, the communication processor can be receiving tuples from another data module to load a different set of  $B_1$ . A final problem occurs when  $p$  is extremely small. In this case,  $B_1$  has to be chosen to be excessively large in order to attain a reasonable join output rate. In this case, it may be better to send different sets of tuples of relation  $A$  to a set of spare data modules, each with a small buffer memory. Relation  $B$  from this data module is then broadcast simultaneously to this set of spare data modules where the join can be performed in parallel. This approach provides a solution for an extremely small value of  $p$  without constraining on the size of  $B_1$ . Buffers  $B_1$  can be kept relatively small at each data module. The use of spare data modules can also be applied for load balancing. This will be discussed in the next section.

TABLE I.—Sample of Values of  $s_2$  for Various Values of  $s_1$  and  $p$

$s_1$	$p$	Simulation				Server Utilization	Estimation upper bound of $s_2$
		mean $s_2$	std. dev $s_2$	Pr(queue size $< s_2$ ) $\leq 0.95$			
5	0.01	0.003	0.055	0	(0.9970)	0.099	0.53
	0.10	0.152	0.445	0	(0.8737)	0.477	0.80
	0.20	10.538	7.784	24	(0.9440)	0.976	$\infty$
	$\infty$	$\infty$	$\infty$	$\infty$	(1.0)	1.0	$\infty$
10	0.01	0.015	0.122	0	(0.9850)	0.191	0.57
	0.02	0.068	0.266	0	(0.9352)	0.377	0.68
	0.03	0.264	0.620	1	(0.9480)	0.570	0.97
	0.04	0.925	1.313	3	(0.9407)	0.773	1.89
	0.05	9.572	6.495	20	(0.9460)	0.978	$\infty$
	$\infty$	$\infty$	$\infty$	$\infty$	(1.0)	1.0	$\infty$
15	0.01	0.032	0.183	0	(0.9698)	0.280	0.62
	0.02	0.252	0.550	1	(0.9570)	0.588	0.98
	0.03	3.066	3.484	10	(0.9588)	0.880	3.80
	0.04	$\infty$	$\infty$	$\infty$	(1.0)	1.0	$\infty$
25	0.01	0.177	0.523	0	(0.8641)	0.486	0.80
	0.02	29.993	12.346	46	(0.9534)	0.996	$\infty$
	0.03	$\infty$	$\infty$	$\infty$	(1.0)	1.0	$\infty$
40	0.01	0.890	1.265	3	(0.9500)	0.770	1.91
	0.02	$\infty$	$\infty$	$\infty$	(1.0)	1.0	$\infty$
50	0.01	22.332	10.911	36	(0.9500)	0.994	$\infty$
	0.02	$\infty$	$\infty$	$\infty$	(1.0)	1.0	$\infty$
75	0.001	0.007	0.082	0	(0.9933)	0.141	0.55
100	0.001	0.013	0.115	0	(0.9865)	0.192	0.57
120	0.001	0.019	0.136	0	(0.9816)	0.229	0.59
140	0.001	0.028	0.169	0	(0.9729)	0.278	0.61
160	0.001	0.046	0.225	0	(0.9578)	0.317	0.63

#### 4. DISCUSSION

The design we have proposed in this paper uses both associative and distributed processing to enhance the system performance. Associative processing is used so that the data can be preprocessed and only the necessary data are selected. The amount of communications is therefore reduced. The associative processors designed are also very simple so



that they can be implemented easily using VLSI technologies. On the other hand, distributed processing is used in the design of the system and therefore many useful techniques in distributed databases can be applied to enhance the performance. These techniques include query processing, file placement and migration, rollback and recovery, etc. We discuss several of these issues here.

#### A. Query processing

Since the system functions as a distributed database, it is possible to optimize the sequence of the processing of queries so that some optimization criteria, such as minimum total time, can be satisfied. There are many techniques developed, e.g. [WON76, WON77, EPS78, HEV79], which can be used to optimize the processing of queries here. The processing of queries using this kind of data flow analysis is new in this design. Previously, the intermediate result of a query usually had to be stored in a temporary file before it could be used, e.g. DIRECT. By allowing intermediate results to be piped to their destinations, higher throughput can be achieved.

#### B. File placement and migration

Because each data module can read only one file at a time, query processing will be sequentialized if all the queries access files on the same data module. By allocating the files on the system so that related files are allocated to the same cluster and files that need to be processed together are allocated to different data modules, conflicts can be kept to a minimum and maximum utilization of resources can be attained. There are many file placement techniques developed, e.g. [CHU69, CAS72, MAH76, MOR77, WAH79] and can be applied to allocate the files on the system. The problem is easier to solve than the general file placement problem because duplicate copies are usually not needed in the system. Further, as the access characteristics change, the files can be reallocated dynamically.

#### C. Concurrency control

Although there may exist multiple copies of a file on the system, we do not allow multiple copies to exist on the same type of memory device. Therefore, multiple copies of a file may exist in devices of different speeds and the copy on the fastest device is considered to be the primary copy. All the other copies are considered as secondary copies and no write throughs are done during updates. With this restriction, the problem on concurrent updates is solved easily because the processor at each data module provides a secure gateway to the data on the device.

#### D. Load balancing

Although our design does not require staging to transfer the files to a fast memory before the processing, it is not

able to utilize multiple processors to process a join in parallel as is done in DIRECT. However, we can provide some spare data modules on the system, each of which has a small buffer memory. Instead of sending tuples directly from one data module to another, different sets of tuples of one relation are sent to the spare data modules, and the tuples from the other relation are broadcast to the spare data modules. The join is therefore performed in parallel in the spare data modules. In this sense, this resembles DIRECT's design. However, there are two significant differences. First, data sent to the spare modules have been reduced in size by the selection processing modules of the two data modules and therefore their sizes are reduced. Second, the distributed system approach allows queries to be processed very efficiently because data can continuously be piped to other data modules as they are produced. A detailed evaluation of this system will be presented in a future paper.

## 6. CONCLUSION

In this paper, we have proposed a database machine design which facilitates query processing and parallel processing. Since the system is hierarchically designed, it can easily be extended to a very large database in the future. Communications at higher levels can be reduced by carefully allocating related files to individual clusters so that most of the communications are intra-cluster. The logic we have designed in the associative processing module is very simple and can be extended modularly. This is different from the conventional approach in which the degree of associative processing is limited by the number of read heads. Further, we have designed the system based on the assumption that current memory technologies are used, e.g. movable arm disk, CCD, etc. This means that the system can be built now. However, we have made no restriction on the structure of the memory which is assumed to be a single loop of data. If, in the future, data can be stored inexpensively in parallel transfer disks, the data can be assumed to be buffered so that a serial stream of access is always achieved. Since the network provides a homogeneous medium for data transfer, heterogeneous memory devices can always be connected together. This advantage also facilitates future evolution of the system. Lastly, our design implements both low level database operations and high level query optimization. The network approach is very versatile and allows high level operations to be implemented.

Our proposed design is given in a functional form. Although individual modules are separated, they do not have to be implemented in individual hardware. In fact, most of these functions can be built in either hardware and software. A detailed evaluation of the system will be shown in a future paper.

## BIBLIOGRAPHY

- [AMP78] Ampex Corporation, PTD-9300, Parallel Transfer Disk Drive, Redwood City, CA. 1978.

- [BAN79] Banerjee, J., Hsiao, D. K., and Kannon, K., "DBC—A Data Base Computer for Very Large Data Bases," *IEEE Trans. on Computers*, Vol. C-28, No. 6, June 1979.
- [BAU76] Baum, R. I. and Hsiao, D. K., "Data Base Computers—A Step towards Data Utilities," *IEEE Trans. on Computers*, Vol. C-25, No. 12, Dec. 1976.
- [BER79] Berra, P. B. and Oliver, E., "The Role of Associative Array Processors in Data Base Machine Architecture," *IEEE Computer*, March 1979.
- [CAS72] Casey, R. G., "Allocation of Copies of a File in an Information Network," AFIPS, SJCC, 1972.
- [CHA78] Chang, H., "On Bubble Memories and Relational Data Base," 4th Int'l Conf. on Very Large Data Bases, Berlin, Sept. 1978.
- [CHU69] Chu, W. W., "Multiple File Allocation in a Multiple Computer System," *IEEE Trans. on Comp.*, Vol. C-18, No. 10, Oct. 1969.
- [COD70] Codd, E. F., "A Relational Model of Data for Large Shared Data Bases," *CACM*, Vol. 13, No. 6, June 1970.
- [COM76] Special Issue in Data Base Management Models, *Computing Surveys*, Vol. 8, No. 1, March 1976.
- [DEW79] DeWitt, D. J., "DIRECT—A Multiprocessor Organization for Supporting Relational Data Base Management Systems," *IEEE Trans. on Computers*, Vol. C-28, No. 6, June 1979.
- [EPS78] Epstein, et al., "Distributed Query Processing in a Relational Data Base System," Report No. UCB/ERL M78/18, Electronics Research Laboratory, University of California, Berkeley, CA., 1978.
- [HEV79] Hevner, A. G. and Yao, S. B., "Query Processing in Distributed Data Bases," *IEEE Trans. on Software Engineering*, Vol. SE-5, No. 3, May 1979.
- [HOL56] Hollander, G. L., "Quasi-Random Access Memory Systems," *AFIPS Conf. Proc.*, EJCC, 1956.
- [KER79] Kerr, D. S., "Data Base Machine with Large Content Addressable Blocks and Structural Information Processors," *Computer*, Vol. 12, No. 3, March 1979.
- [LAN79] Langdon, Jr., G. G., "Data Base Machine, An Introduction," *IEEE Trans. on Computers*, Vol. C-28, No. 6, June 1979.
- [LIN76] Lin, C. S., et al., "The Design of a Rotating Associative Memory for Relational Data Base Applications," *ACM Trans. on Data Base Systems*, Vol. 1, No. 1.
- [LIP78] Lipovski, G. J., "Architectural Features of CASSM: A Context Addressed Segment Sequential Memory," *Proc. 5th Ann. Symp. on Comp. Arch.*, ACM-SIGARCH.
- [LIT61] Little, J. D. C., "A Proof of the Queuing Formula,  $L = \lambda w$ ," *Operations Research*, 9, 1961.
- [MAH76] Mahmoud, S. and Riordon, J. S., "Optimal Allocation of Resources in Distributed Information Networks," *ACM Trans. on Data Base Systems*, Vol. 1, No. 1, March 1976.
- [MAR68] Marshall, K. T., "Some Relationships between the Distributions of Waiting Time, Idle Time and Input/output Time in the GI/G/1 Queue," *SIAM Journal of App. Math.*, 16, 1968.
- [MOR77] Morgan, H. L. and Levin, K. D., "Optimal Program and Data Locations in Computer Networks," *CACM*, Vol. 20, No. 5, May 1977.
- [OZK77] Ozkarahan, E. A., et al., "Performance Evaluation of a Relational Associative Processor," *ACM Trans. on Data Base Systems*, Vol. 2, No. 2, June 1977.
- [PAR72] Parhami, B., "A Highly Parallel Computing System for Information Retrieval," *AFIPS Conf. Proc.*, 1972, FJCC, Vol. 41, part II.
- [RAM78] Ramamoorthy, C. V., Turner, J. L., and Wah, B. W., "A Design of a Cellular Associative Memory for Ordered Retrieval," *IEEE Trans. on Computers*, Vol. C-27, No. 9, Sept. 1978.
- [SCH79] Schuster, S. A., et al., "RAP.2—An Associative Processor for Data Base and its Applications," *IEEE Trans. on Comp.*, Vol. C-28, No. 6, June 1979.
- [SLO70] Slotnick, D. L., "Logic Per Track Devices," *Advances in Computers*, Academic Press, 1970.
- [SU79] Su, S. Y. W., et al., "The Architectural Features and Implementation Techniques of the Multi-cell CASSM," *IEEE Trans. on Computers*, Vol. C-28, No. 6, June 1979.
- [WAH79] Wah, B. W., "A Systematic Approach to the Management of Data in Distributed Data Bases," Ph.D. Dissertation, University of California, Berkeley, 1979.
- [WON76] Wong, E. and Youssefi, K., "Decomposition—A Strategy for Query Processing," *ACM Trans. on Data-bases*, Vol. 1, No. 3, Sept. 1976.
- [WON77] Wong, E., "Restructuring Dispersed Data from SDD-1: A System for Distributed Data Bases," Comp. Corp of America Tech. Rep. CCA-77-03, 1977.



## **Distributed Data Base Processing**

Distributed Data Processing (i.e., the sharing of computing resources by several processors, each having a specified task to perform; and which may be either centralized or geographically distributed), is one of the more exciting developments since the appearance of the transistor in 1948.

Distributed Data Processing encompasses data communication, teleprocessing, and data base technology. Designing and using distributed data processing systems have created a need to solve a wide range of problems, including problems associated with expanding memories, operating system capabilities and data sharing.

The sessions in this technical area are concerned with distributed data base technology. The focus is on current practices and proposed new solutions for problems unique to the field.

Alyce Jackson, Ph.D, the technical area coordinator, has tailored each session to address a particular issue. The speakers are of diverse backgrounds and areas of expertise; each will discuss their involvement and contribution to areas of distributed data base technology.

Dr. Baharat Bhargava will chair a session that addresses the issues of concurrency, consistency, and reliability. Dr. Ed Lee of TRW in Redondo Beach, California will present speakers discussing issues pertaining to distributed architecture design. Dr. A. Jackson will chair a session that addresses topics such as concurrency coordination, federated data base systems, and computer chaining. The final session in this technical area presents a survey of experiences with various data base management systems.



*Alyce Jackson  
Area Director*



# System deadlocks resolution

by KOJI NEZU

Nippon Electric Co., Ltd.  
Kawasaki, Japan

## INTRODUCTION

With rapid large online systems growth, the system deadlocks problem is becoming of major importance. Sharing on increasingly sophisticated set of services and resources among an increasing number of terminal users leads to an increase in the chance for deadlock occurrence.

Two approaches for the problem are: (1) deadlock occurrence prevention or avoidance; (2) early deadlock detection and quick recovery.

Approach 1 is ideal, if it would be completely realized within an allowable system performance degradation limit. However, a system design to fulfill this requirement is very difficult in most online systems. Typical is multiaccess database. For this reason, approach 2 must be studied and a practical method that realizes fully automatic deadlock detection and recovery must be developed.

A general deadlock detection algorithm has been known.<sup>1</sup> Moreover, high speed hardware which simulates the algorithm has been proposed.<sup>2</sup> However, no general algorithm has been developed for recovery.

Recovery from system deadlocks is carried out by aborting appropriate deadlocked processes, freeing the resources that had been exclusively allocated to these processes and making it possible to run the other deadlocked processes whose execution has been blocked by a round robin wait for resources among deadlocked processes.

As an example, assume two processes,  $Pa$  and  $Pb$ , are detected to be mutually deadlocked. It is obvious that there are two solutions for resolving this problem, one is process  $Pa$  abortion, and another is process  $Pb$  abortion. When there are more than two deadlocked processes, the existence of multiple solutions may also be expected. In fact, as proven in the following section, multiple solutions always exist.

The above discussion indicates that the following three steps are necessary for recovery: (step 1) obtain solutions; (step 2) select one solution to actually aborted processes and execute abort; (step 3) restart aborted processes at appropriate instants.

In order to accomplish an optimum decision at step 2, all the solutions should be known for step 1.

This paper presents a new systematic method for obtaining all the solutions for step 1.

Further discussion about steps 2 and 3 does not appear in

this paper. The papers indicated in the references<sup>1,3</sup> will be helpful for understanding these subjects.

## PRELIMINARIES

To clarify the discussion, consider the following system model, which was proposed by Habermann.<sup>4</sup>

A set of sequential processes which share system resources in such a way that, while resource  $A$  is allocated to process  $Pi$ , no other process  $Pj$  can seize  $A$ , and an allocated resource is not released until it has fulfilled its task.

Generally every resource is not necessarily unique. Therefore, consider such a system that consists of  $n$  processes  $P_1, P_2, \dots, P_n$  and  $m$  different kinds of resources ( $n \geq 1, m \geq 1$ ).

Although deadlock detection is not the subject of this paper, a brief description about the detection algorithm is also necessary.

The deadlock detection algorithm given by Shoshani and Coffman is summarized as follows.<sup>1</sup>

The first step is to pick up processes which are requesting no specific resources or requesting only free resources. It is clear that these processes are not deadlocked. The second step is to assume the resources allocated to the picked up processes as "free resources," and go back to the first step. By this loop, all the non-deadlocked processes are picked up. Therefore, if all the processes are confirmed to be non-deadlocked, no deadlock exists in the system. Otherwise, deadlock exists and remaining processes are deadlocked.

## SOLUTIONS

It can be said that the deadlocked state complexity level is closely related to the number of deadlocked processes. When there are two deadlocked processes, all solutions are immediately known. As illustrated in the preceding section, that is  $Pa$  and  $Pb$  are deadlocked mutually, solutions are  $\{Pa\}$  and  $\{Pb\}$ . However, when there are more than two deadlocked processes, probably all the solutions cannot be obtained immediately. Up to now, the term 'solution' has been used in a tacit sense. Now, let us define it clearly as follows:

"A solution is the smallest subset of deadlocked processes, whose aborting resolves the deadlock."

For example, consider that the deadlock in which four processes  $P_1, P_2, P_3$  and  $P_4$  fall, and that, by some means, it is confirmed that, by aborting both  $P_1$  and  $P_2$ , deadlock is resolved, but that by aborting only  $P_1$ , or  $P_2$ , the deadlock is not resolved. Then,  $\{P_1, P_2\}$  is a solution. Obviously by aborting  $\{P_1, P_2, P_3\}$  it is resolved, but abortion of  $P_3$  in addition to  $\{P_1, P_2\}$  is meaningless. Above definition eliminates  $\{P_1, P_2, P_3\}$  from solutions.

Consider in the same example,  $\{P_2, P_3, P_4\}$  is aborted. Obviously, the deadlock is resolved because the only remaining process is  $P_1$ . As  $P_1$  is an element of above solution, this means another solution, which does not involve  $P_1$ , exists. From this example, the following general theory is induced.

“When deadlock occurs, multiple solutions always exist.”

OBTAINING A SOLUTION

Among the multiple solutions, one solution can be obtained by the following procedure;

Procedure 1

- Step 1: Arrange all the deadlocked processes linearly in an arbitrary order, such as  $(Pd_1, Pd_2, \dots, Pd_n)$ .
- Step 2: Scan from  $Pd_1$  to  $Pd_n$ , marking such processes that must be aborted for deadlock resolution, if no un-marked processes, those that have been scanned, are aborted.
- Step 3: Collect all the marked processes. This set is one of the solutions.

It may be easily understood that, by aborting the above obtained set of processes, deadlock is resolved, but that by aborting any true subset, it is not resolved. Therefore, the set of above marked processes is a solution.

Note that a solution can be obtained in a straightforward way.

Example 1

Obtain a solution for the system deadlock shown in Table I.

TABLE I.—System State Table

RESOURCE NAME		R <sub>1</sub>	R <sub>2</sub>	R <sub>3</sub>	R <sub>4</sub>
NUMBER OF RESOURCES		2	3	3	2
NUMBER OF RESOURCES ALLOCATED/REQUESTED*	P <sub>1</sub>	1/0	1/0	0/3	0/1
	P <sub>2</sub>	0/2	1/0	1/0	0/1
	P <sub>3</sub>	0/0	1/0	0/2	0/0
	P <sub>4</sub>	0/0	0/0	0/0	2/0
	P <sub>5</sub>	1/0	0/3	2/0	0/1

\* requested but not allocated to the process

By applying Shoshani and Coffman’s algorithm explained in the previous section, it is known that, except for  $P_4$  which is requesting no resource, all other processes are mutually deadlocked.

Using procedure 1, a solution for the deadlock resolution can be obtained.

- Step 1:  $(P_1, P_2, P_3, P_5)$
- Step 2:  $P_1$ : not marked. ∴The deadlock is resolved by aborting  $P_2, P_3, P_5$ .
- $P_2$ : marked. ∴Even if  $P_3$  and  $P_5$  are aborted, deadlock is not resolved. (Use Shoshani and Coffman’s algorithm hereafter.)
- $P_3$ : not marked. ∴The deadlock is resolved by aborting  $P_2$  and  $P_5$ .
- $P_5$ : marked. ∴The deadlock is not resolved by aborting  $P_2$ .
- Step 3: one solution =  $\{P_2, P_5\}$

Note that the solution depends on the process order arranged in step 1. Therefore, denote each solution, suffixing the process order, such as  $\alpha_{(1,2,3,5)} = \{P_2, P_5\}$ .

ALL SOLUTIONS

Another solution for the above example can be obtained by placing one of the processes contained in the above solution  $\{P_2, P_5\}$  to the left end of the line in step 1, such as  $(P_2, P_1, P_3, P_5)$ , and then executing step 2. Obviously, the new solution, that is  $\alpha_{(2,1,3,5)}$ , is different from the previous solution, because  $\alpha_{(2,1,3,5)}$  does not contain  $P_2$ . For the third and further solutions, the same basic idea can be applied. However, existence of more than two solutions is not guaranteed. Therefore, before applying procedure 1, confirming step is necessary.

The following theorem is useful for this objective.

“If there is any set of deadlocked processes which, for every already known solution, contains at least one same process, where the deadlock can be resolved without aborting this set, at least one new solution exists. Otherwise, no new solution exists.”

The first half may be trivial. The latter half is proven as follows: From the definition of “solution,” a solution is neither subset nor superset of another solution. Therefore, a new solution, when it is compared with an already known solution, does not contain at least one deadlocked process that is contained in the already known solution. This relation is maintained over all already known solutions.

Example 2

Obtain all the solutions for the system state shown in example 1.

The first solution has been obtained.

$$\alpha_{(1,2,3,5)} = \{P_2, P_5\}$$

The second solution is obtained by placing  $P_2$  on the left end of the line in step 1 of procedure 1.

$$\alpha_{(2,1,3,5)} = \{P_1, P_5\}$$

For the third solution, its existence must be checked. To apply the theorem, a check table is convenient.

Note that in Table II, all the combinations, which involve a single element picked up from each already known solution, are checked. The reason will be clear later.

From the table, it is confirmed that a new solution exists and does not contain  $P_5$ .

Placing  $P_5$  at left end of the line, a new solution is obtained by procedure 1.

$$\alpha_{(5,1,2,3)} = \{P_1, P_2, P_3\}$$

Existence of the fourth solution is checked in the same way. However, a direct check about all the combinations for the three already obtained solutions is not necessary, because all the combinations up to the second solution have been checked in Table II. An additional check must be made for the 'YES' row in Table II, that is  $P_5$  row, and the third solution.

As all rows are NO, no further solution exists. Summing up, all the solutions are

$$\{P_2, P_5\}, \{P_1, P_5\}, \{P_1, P_2, P_3\}$$

### CHECK TABLE NORMALIZATION

In this section, redundant rows elimination from check tables is discussed.

Clearly, the objectives for making check tables is to pick up all the subsets of the deadlocked processes which contain new solutions. From this objective, it may be understood that the second and the third rows in Table II are redundant, because, in these rows, the subset for the fourth row are checked. More concretely speaking, at the second and third rows, solution existence in  $\{P_1, P_3\}$  or  $\{P_2, P_3\}$  respectively, is checked. At the fourth row, the solution existence in  $\{P_1, P_2, P_3\}$  is checked. If the fourth row is "YES," it is not necessary to check the second and the third row due to the above mentioned objectives. If it is "NO," the latter two

TABLE II.—Third Solution Check Table

DLP*s not being aborted	Can deadlock be resolved?
$P_1, P_2$	NO
$P_2, P_5$	NO
$P_1, P_5$	NO
$P_5$	YES

\*: Deadlocked process

↑ all combinations extracting one process from each solution.

TABLE III.—Fourth Solution Check Table.

DLPs not being aborted	Can deadlock be resolved?
$P_1, P_5$	NO
$P_2, P_5$	NO
$P_3, P_5$	NO

(Checked in Table 1)

rows are obviously "NO" too. Therefore, no check is necessary at second and third rows.

In summary, check table simplification is accomplished by eliminating all rows which are supersets of another row in the same table.

This simplification is called "check table normalization."

By applying "normalization," the procedure shown in the example 2 can be executed more clearly and efficiently.

Normalization effectiveness becomes conspicuous for more complicated deadlocks.

### Example 3

Assume, in example 1, that a new request for three  $R_2$  resources is issued from  $P_4$ , and  $P_4$  also falls into deadlock, as a result.

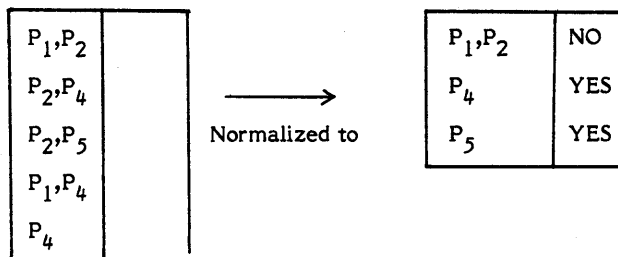
The first solution is

$$\alpha_{(1,2,3,4,5)} = \{P_2, P_4, P_5\}$$

The second solution is

$$\alpha_{(2,1,3,4,5)} = \{P_1, P_4, P_5\}$$

Check table for the third solution is:



$P_4, P_5$	
$P_1, P_5$	
$P_5$	

Third solution:

$$\alpha_{(4,1,2,3,5)} = \{P_1, P_2, P_5\}$$



Check table for the fourth solution is:

P <sub>1</sub> ,P <sub>4</sub>	
P <sub>2</sub> ,P <sub>4</sub>	
P <sub>4</sub> ,P <sub>5</sub>	
P <sub>1</sub> ,P <sub>5</sub>	
P <sub>2</sub> ,P <sub>5</sub>	
P <sub>5</sub>	

→ Normalized to

P <sub>1</sub> ,P <sub>4</sub>	NO
P <sub>2</sub> ,P <sub>4</sub>	NO
P <sub>5</sub>	YES

Fourth solution:

$$\alpha_{(5,1,2,3,4)} = \{P_1, P_2, P_3\}$$

Check table for the fifth solution is:

P <sub>1</sub> ,P <sub>5</sub>	NO
P <sub>2</sub> ,P <sub>5</sub>	NO
P <sub>3</sub> ,P <sub>5</sub>	NO

(Normalized from origin)

No further solution exists.

### CONCLUSION

To accomplish non-stop operation of large online systems, a new general algorithm, for obtaining deadlock resolution solutions, has been proposed. Experimental implementation on a multiaccess database system is planned at Central Research Laboratories, NEC.

### ACKNOWLEDGMENT

The author would like to thank H. Kubo and M. Mizuma, both of Central Research Laboratories NEC, for their critical advice in preparation of this paper.

### REFERENCES

1. Shoshani, A. and Coffman, E. G., "Prevention, detection, and recovery from system deadlocks," *Proc. 4th Annual Princeton Conf. on Information Science and Systems*, March 1970.
2. Nezu, Koji, "Simplified deadlock detector," *Trans. IECE*, Vol. 61-D, No. 9, 1978 (in Japanese).
3. Verhofstad, Joost S. M., "Recovery techniques for database systems," *Computing Surveys*, Vol. 10, No. 2, June 1978.
4. Habermann, A. N., "Prevention of system deadlocks," *Comm. of the ACM*, Vol. 12, No. 7, July 1969.

# Database semantic integrity for a network data manager\*

by ELIZABETH FONG and STEPHEN R. KIMBLETON

National Bureau of Standards  
Washington, D.C.

## 1. INTRODUCTION

The goal of semantic integrity is assuring that data within a database is logically correct. Logical correctness is evaluated by showing that the data represents a valid abstraction or model of a 'real world' application. This is required for effective use of the database.

The starting point in assuring database semantic integrity is a clean (semantically correct, i.e. logically correct) database and the objective is ensuring that subsequent updates also result in a clean database. Since individual updates may involve operations across data structures, may require several statements of the data manipulation language being used, and may involve logically interrelated data, assuring semantic integrity is a difficult problem which currently lacks a complete solution.

This paper has two major objectives. The primary objective is establishing a collection of capabilities appropriate for a semantic integrity system. This assumes an environment providing a uniform mode of access for the network user to multiple, remote heterogeneous DBMSs. The secondary objective is providing a brief survey of the existing semantic integrity literature. Thus, the following two sections explore the existing approaches to semantic integrity. Thereafter, the environment supporting remote database access is described. This is followed by a discussion of the design of an Experimental Semantic Integrity System (XSIS) currently being constructed at the National Bureau of Standards.

## 2. WHAT IS SEMANTIC INTEGRITY?

Data within a DBMS can be in error for one of three reasons: incorrect entry, system failure, or conflict with the intended meaning of the data. Incorrect entry occurs when a *wrong value gets keyed-in* during the data entry process.

For example, due to a keypunch error, a person's weight might be entered as 87 rather than 78 kgs.

System errors reflect a variety of failures of design or implementation including: security violations, hardware failures, and failures in system or DBMS software such as the concurrency control mechanism.

Conflicts with the intended meaning of data occur through violations of perceived interrelationship among the data within DBMS. For instance, the sum of division capital expenditure budgets, after an update, may no longer equal the organizational capital expenditure budget.

Semantic integrity is concerned with assuring that such violations cannot occur. Since such violations represent a conflict with the understood meaning of data, and since this understood meaning is usually not formally expressed, the goal of achieving semantic integrity has proved elusive.

Because semantic integrity is concerned with the logical meaning of data, it can also provide limited assistance in detecting data entry errors or system failures. For instance, through value-range specification, the semantic integrity system could detect the error in entering a person's weight as 570 kg. rather than 70 kg. It could not detect the difference between 78 and 87 kgs. Similarly, system failures resulting in major data errors can be detected when the data is examined.

### 2.1 Approaches to semantic integrity

There are two primary approaches to semantic integrity. The assertion based approach permits users or Database Administrators to specify those semantic integrity rules which seem important. This approach facilitates incremental incorporation of a semantic integrity capability for a DBMS.

The disadvantages of the assertion based approach are potentially significant and include: i) the inability to determine the completeness of a collection of rules since there is no basic point of reference for their assessment, ii) the possibility that adding a new rule will show that a database previously thought to be semantically correct now suffers from integrity violations, and iii) the need for explicit incorporation of consistency checks on the collection of rules, which might lead to significant overhead.

The major alternative to the assertion based approach depends on a thorough development and application of the

\* 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 30602-77-0068. 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 for the purpose.

concept of type [BRODM 78]. Strong data types are constructed from system data types. Moreover, through exploiting the concepts of aggregation and generalization, type interrelationships can be formally specified. Such specification has the advantage of permitting semantic integrity assertions to be checked *statically at compile-time rather than run-time*. Additionally, the completeness and consistency of data type specifications can be shown.

The disadvantage of this approach is the explicit requirement for complete specification. For databases of significant size, completion of the specification process requires a major investment of effort. Another drawback is that not all semantic integrity requirements can be specified using the data type concept because certain validity criteria are "value" rather than "type" oriented. A combined approach may prove best in an operational environment.

## 2.2 Semantic integrity system

A combined approach to supporting semantic integrity seems appropriate. Implementing such a system requires specification of three major components: i) the rules specifying the semantic integrity constraints which may be either type or value constraints, ii) the process for checking conformance with these rules, and iii) the actions which occur upon detection of an integrity violation.

### 2.2.1 Rules specifications

The semantic integrity rules or constraints need to be stated prior to database use. These constraints specify all the required information to be used during rules enforcement time. Strong data type specifications effectively augment the traditional concept of schema and are stated through denotational or declarative methods at data definition time. Assertion based approaches require a means for specifying the assertion and are evaluated while a request is being processed.

Designing a semantic integrity specification language requires considering two main issues: i) the style of the specification language used by users and database administrators, and ii) the volume of information required by the system to check conformance with the rules.

### 2.2.2 Rules enforcement

Evaluating conformance with semantic integrity rules can be thought of as being performed by an abstract observer or daemon monitoring database operations. The times at which the daemon can observe the database are defined; there may be times during an update when the daemon is precluded from judging whether semantic integrity is being maintained.

Implementation of the daemon requires considering three major issues: i) the types of tests which the daemon may

perform, ii) the information required to support these tests, and iii) the cost of performing them. This last item is very important from a practical point of view.

### 2.2.3 Violation actions

Detecting a semantic integrity violation requires flagging the error and, probably, rejecting the update. The precise rule(s) responsible for the violation action should be identified. The options may be a call to a user-specified error routine, reporting an error message to the user, or semi-automated error elimination by the system.

## 2.3 Related works on semantic integrity

Much of the existing semantic integrity literature is concerned with isolated aspects of the global problem—placement of responsibility, data semantics, specification languages, invocation techniques, and supporting system environments. For instance, [FERNE 76], [GRAVR 75], and [MACHC 76] all describe high level semantic integrity specification languages. In [WEBEH 76] semantic integrity is viewed in the context of state transitions; therefore, constraints are expressed upon database operations. Buneman and Morgan [BUNEO 77] developed "alerting" mechanisms for supporting semantic integrity. Several selected semantic integrity systems are reviewed below.

### 2.3.1 Brodie

Brodie's approach [BRODM 78] views semantic integrity as a (semantic integrity) system rather than user responsibility. A specification language together with a verification methodology is developed. The specification language is based on a denotational approach and the emphasis is on proving consistency and completeness.

Since Brodie's approach is not currently implemented, support system requirements have not been identified. However, the extremely systematic approach which is presented requires complete specification at database design time. Operational use would require investigation of the issues of sizing and flexibility.

### 2.3.2 McLeod

McLeod's approach [MCLED 76, HAMMM 75] is assertion based, views semantic integrity as a system responsibility, and also describes a semantic integrity system. Special emphasis is placed on the design of a constraint specification language for a relational data model. The specification language is based on using a high level, non-procedural language permitting specification of: i) constraints, ii) times at which the constraints are to hold, and iii) actions to be taken on occurrence of a constraint violation.

### 2.3.3 System R

System R, a relational database management system developed at IBM, provides semantic integrity facilities as part of the SEQUEL language [ESWAK 75]. The approach uses the assertion based method. An assertion can be any SEQUEL predicate evaluating to a Boolean. System R provides a very extensive collection of supporting capabilities. Various types of semantic integrity features are provided, such as: tuple and set assertions, state and transition assertions, and immediate and delayed assertions. Semantic integrity enforcement is implemented using system triggers and, if the result of a transaction is proven to violate an assertion, the transaction is rejected and a predefined procedure or failure action is invoked.

### 2.3.4 INGRES

INGRES is a relational database management system developed at University of California, Berkeley. Stonebraker [STONM 75] introduced semantic integrity assertion statements in the INGRES system as one or more range statements plus an integrity qualification. These assertion statements in the form of predicates are appended to all user interactions with a database. Thus certain types of update errors are prevented. This implementation technique has been referred to as query modification, and has been characterized as easy to implement. The INGRES approach of incorporating semantic integrity via query modification minimizes the requirement for support. However, it also limits the set of semantic integrity features which are provided.

### 2.3.5 CODASYL

The CODASYL Data Description Language Committee Journal of Development specification [CODAS 79] provides a user-written procedure invocation mechanism which allows integrity requirements to be programmed in procedural code by the user. The keyword CHECK with user specified parameters provides the triggering mechanism when a data item is changed.

CODASYL does not use the system approach but, rather, places the burden of specification and enforcement of semantic integrity upon users who must write application programs.

The CODASYL specification supports an underlying network data model with stringent structural requirements; therefore, facilities for duplicate checks, member record insertion conditions, and unique keys checks are supported.

[MELOR 79] proposes semantic integrity facilities to be incorporated in a CODASYL-like DBMS. These integrity constraints are meant to enhance those integrity capabilities that are provided by the CODASYL specification and those that are inherent in the network data model.

## 3. SEMANTIC INTEGRITY FEATURE LIST

The above brief survey reveals that, currently, there is no unified, practical, comprehensive and complete approach to database semantic integrity. Semantic integrity features offered within a DBMS range from very simple data type checking to complex assertion processing. We present a gross feature list with the following categorizations:

### 3.1 Strong data type constraints

Semantic integrity constraints based on an extension of the data type concept can be specified at data definition time. Such constraints are related to the concept of strong typing developed for program languages.

The concept of a strong data type arose from developments in abstract data types [LISKB 74; GUTTG 77] which can be informally described as special purpose entities with constrained usage properties. Constraints typically include both the operations which may be performed on a given data element as well as the collection of other data elements which may be involved in binary operations.

Some examples of strong data type constraints are described as follows:

1. Value constraints—specify the range of acceptable values, establish whether a value must be specified, and define whether a data value must be unique. For instance, data values may be required to lie within certain bounds (e.g., age between 1 to 100). Data values may also be constrained to an enumerated set (e.g., colors are red, green, blue). Data values may be specified to be essential or non-missing in which case missing values are considered to be semantically incorrect (e.g., data element EMPNO must not be missing). Data values may be required to be unique (e.g., EMPNO must be unique).
2. Extended format constraints—permit specification of a data format pattern composed from primitive types, such as character, integer or real. For example, the first character of a supplier number may be required to be the letter *S*.
3. Domain compatibility—supports assurance that cross domain operations, e.g., binary operators, are applied against compatible units and prohibits such operations against incompatible domains. For example, automatic invocation of scaling factors is required when some weights are expressed in kilograms and others in pounds. Some automatic techniques for performing such conversions are given in [KARRM 78]. Invalid comparisons, e.g., comparing weight to time, should be flagged as constraint violations.
4. Legal operation constraints—limit the operations which can be performed on a given domain to those judged semantically correct. Thus, a set of legal operations may be associated with a data item. Attempting to per-

form any other operation will result in a semantic integrity violation, e.g.,  $SSN * 2$  or  $NAME > "123"$  will be detected as semantically incorrect.

### 3.2 Constrained data dependency assertions

Integrity constraints may be imposed upon an individual data element or group of data elements. The constraint expresses a condition or predicate on the subset of the database to which the constraint applies. Such constraints across data elements can be further categorized:

1. Data grouping constraints—reflecting logical relationships among data elements. For example, if the data is represented in tabular form, there can be a constraint that is dependent upon the other values in the same row (row constraints), or column (column constraints), or table (table constraints), or collection of tables (inter-table constraints). The constraint: salary of employee must be less than salary of manager, would be a table constraint if salary information and salary of manager appeared in the same table, and an inter-table constraint if this information appeared in two different tables.
2. Aggregate function constraints—based upon built-in aggregate functions such as average, minimum or maximum. For example, average salary may be constrained to be less than \$15,000.
3. Data model constraints—based upon the particular characteristics of a data model such as relational, hierarchical or network.

### 3.3 Transition constraints

Transition constraints specify relation between old and new states of the databases and are invoked when the database changes from one state to another. The two major types of transition constraints are:

1. Old/New transition constraint—During an update operation, there exists an "old" value to be changed to a given "new" value. For example, new salary must be greater than old salary.
2. Nonexistence/existence transition constraint—The insertion operation involves a nonexistence to existence transition, while a deletion involves an existence to nonexistence transition. For example, deletion of an account number may require that the balance be zero.

### 3.4 User-controlled enforcement

Using one of the constraints specified above requires establishing when the constraint is to be invoked. User controlled enforcement permits the user to state WHEN to enforce the integrity constraints:

1. Deferred/immediate enforcement—for transactions requiring more than one data management request, will

permit suspension of integrity constraints until all requests have been issued. The user is responsible for specifying deferred enforcement and the system must be able to back-out the transaction when deferred enforcement results in a constraint violation.

2. ON/OFF enforcement—permits the user to switch integrity checking ON or OFF depending upon the level of integrity needed for the application. It is useful since some integrity checking is costly.

### 3.5 Integrity specification and maintenance facilities

Some integrity checking, such as simple data type checking, is easily supported. More complex kinds of assertions need to be stated, maintained, and invoked at the appropriate time. Some semantic integrity system facilities might be:

1. User-written Application Program Interface—The DBMS does not provide a centralized semantic integrity subsystem but provides interface mechanisms so that users can code their own integrity enforcement routines as an application program. This application program interface is usually available in most of the commercially available DBMS.
2. Integrity Specification Language—The semantic integrity system permits users to specify integrity constraints in a higher-level language. This language is compiled into procedures which are triggered for the enforcement of the constraints.
3. Integrity Constraint Maintenance—The semantic integrity system permits users to read, modify, create and destroy integrity constraint assertions.

### 3.6 Feature summarizations

Not all of the features identified above are offered by each of the reviewed systems. Moreover, substantial differences exist in the implementation of features common to two or more systems. Brodie's approach has not yet been implemented. McLeod's has been partially implemented. System R has not been released as a product. INGRES is available and is being used at several sites. CODASYL specifications are still in the process of enhancement. Implemented versions of CODASYL-like systems usually include the CHECK clause plus interface mechanisms for user-written procedures so that necessary integrity features may be coded by the user via application programs.

The semantic integrity feature list contained in Figure 1 together with the various enforcement approaches provides a basis for the design of a prototype Experimental Semantic Integrity System.

## 4. X SIS SUPPORT ENVIRONMENT

A prototype Experimental Semantic Integrity System (X SIS) is currently under construction at the National Bu-

<b>FEATURES</b>
<b>STRONG DATA TYPE CONSTRAINTS</b>
1. VALUE CONSTRAINTS
2. EXTENDED FORMAT CONSTRAINTS
3. DOMAIN COMPATIBILITY
4. LEGAL OPERATION CONSTRAINTS
<b>CONSTRAINED DATA DEPENDENCY ASSERTIONS</b>
1. SINGLE DATA
2. GROUPS OF DATA
3. AGGREGATE FUNCTIONS
4. DATA MODEL
<b>TRANSITION CONSTRAINTS</b>
1. OLD/NEW
2. NONEXISTENCE/EXISTENCE
<b>USER-CONTROLLED ENFORCEMENT</b>
1. DEFERRED/IMMEDIATE
2. ON/OFF
<b>SPECIFICATION &amp; MAINTENANCE FACILITIES</b>
1. USER-WRITTEN PROGRAMS
2. INTEGRITY SPECIFICATION LANGUAGE
3. CONSTRAINTS MAINTENANCE

reau of Standards. X SIS is designed to operate as part of a distributed, networked environment supporting access to multiple, remote, heterogeneous DBMSs which is provided by an Experimental Network Data Manager (XNDM) [KIMBS 79]. Since this paper explores the issues in defining, implementing and maintaining X SIS based on XNDM support, it is important to have an understanding of its basic structure and interrelationship to the network accessible DBMSs.

#### 4.1 Experimental network data manager (XNDM)

The basic structure of XNDM is illustrated in Figure 2. It is implemented in the C language on a PDP-11/45 attached to the Arpanet and running the UNIX operating system. Server modules exist on target systems to provide local support. XNDM provides a uniform, table based virtual view of data maintained by independent, network accessible DBMS. It differs from a distributed database because: i) the view of data provided the network user is virtual, ii) the DML provided the network user differs from that actually employed by the target DBMSs, and iii) different systems can use different DBMSs, data models, and DMLs.

##### 4.1.1 XNDM data structures

The XNDM data structures are assumed to be constructed by a committee of Data Base Administrators who, collectively, identify the data whose access is to be supported and specify the access paths used to access this data. Although the virtual view of data presented by these structures is relational, i.e., table based, the data models used by the target systems can be relational, hierarchical, or CODASYL.

##### 4.1.2 XNDM data language

The data language for XNDM is termed the Experimental Network Data Language (XNDL). It consists of three major components: a data definition language defining the tables and their data attributes, a data control language providing nondiscretionary access controls and semantic integrity specifications, and a data manipulation language for issuing queries and updates. The DML is based on a subset of SE-QUEL (redundant predicates, sort operations, and a programming language interface are excluded) which has been extended to provide primitives appropriate to specifying and controlling concurrent access to multiple databases. A more complete specification of XNDL is contained in [KIMBS 79].

##### 4.1.3 Interfacing to the target systems

Using XNDM requires the DML statements (XNDL queries or updates) to be translated to the DML employed by the

Figure 1—Semantic integrity feature list.

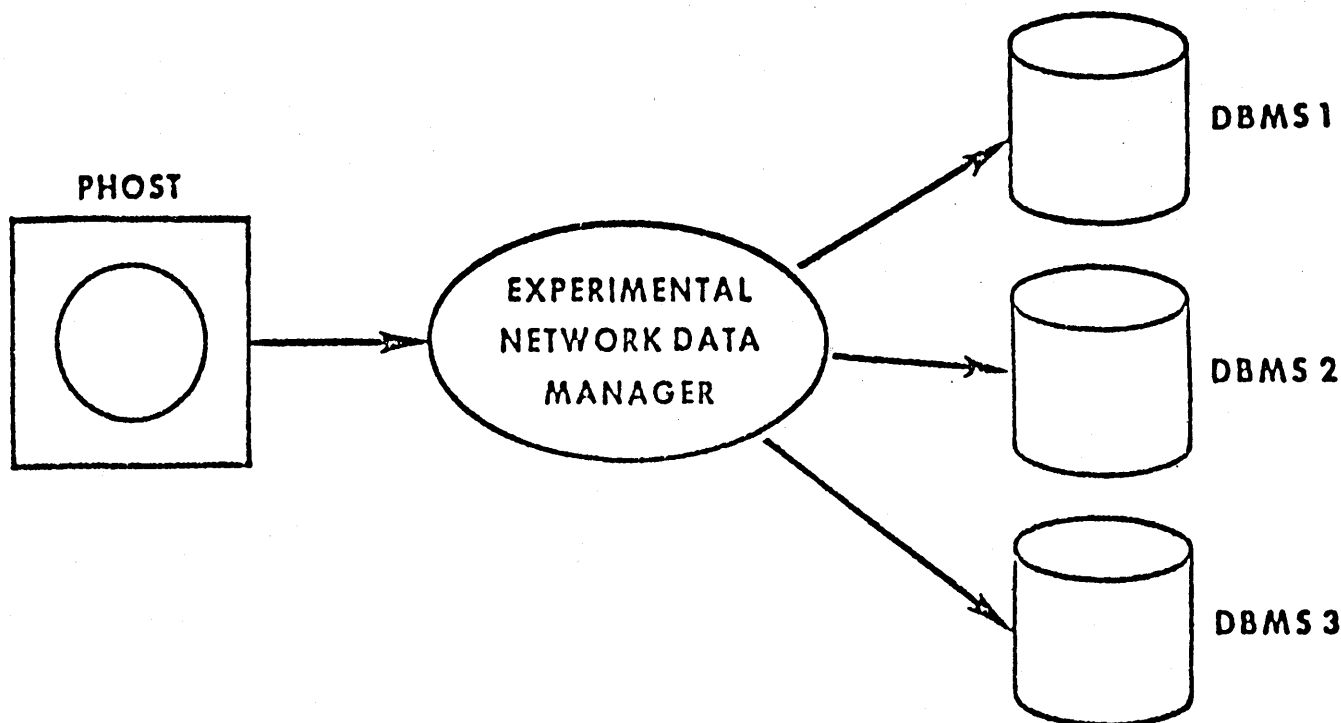


Figure 2—XNDM interface between user program and multiple remote DBMSs.

target system(s). Since this translation process is dependent upon both the preestablished source data structures as well as the locally determined target data structures, it proves substantially harder than that required to support database front ends. Substantial work has, however, been done on XNDM query translation [KIMBS 79], [WANGP 80].

Currently, XNDM does not support updates. A major technical problem in their support is related to the recognized problem of updating views. A general solution to this problem appears difficult if not impossible [DAYAU 78] because of hidden data. Although this is valid for a general, and hence arbitrary view, it may be resolvable in the context of XNDM applications by suitably defining the data structures presented to the user. Such definition would require classification of DBMS data into independent groups coupled with a requirement that a user be able to update all data elements of the group if the user can update any data element of the group. Since remote users often have specialized interests, this approach may be commensurate with implicit organizational requirements.

Deferral of XNDM update capabilities reflects the desire to complete implementation of query support since a thorough understanding of this problem is required to provide the basic addressability information required to support updates. In view of the strong interest which has been expressed to the authors about the impact of the network user on the quality of data maintained by a DBMS, it seemed appropriate to establish the types of guarantees which can be provided together with the problems implicit in their support. This is the objective of this paper.

#### 4.1.4 Need for semantic integrity in XNDM

The preceding suggests that local acceptance of a remote updating capability may well depend upon the extent to which suitable correctness guarantees can be provided. Two basic support mechanisms for providing such correctness can be developed. The first is an appropriate collection of discretionary access controls for assuring that individual remote users can only access data appropriate to their access rights. The second is a semantic integrity support capability. Issues related to the first have been discussed in [WOODH 79]; the remainder of this paper is concerned with the second topic.

#### 4.2 Semantic integrity in the networking environment

Semantic integrity in the XNDM environment differs in several essential ways from that usually associated with centralized DBMSs.

1. Global and local integrity constraint conflicts—reflecting the possibility of a conflict between globally established constraints and those established by local DBMS management.
2. Conflicting assertions—constraints appropriate to different target DBMSs may also be in conflict.
3. Partial data problem—local semantic integrity assertions may involve data not accessible to the network user. Thus, it may prove impossible to provide global

checking which is complete from the local point of view.

## 5. XSYS DESIGN AND IMPLEMENTATION ISSUES

XSYS is also being implemented in C on the same system on which XNDM is being implemented. Its design and implementation are intended to support exploration of both the assertion based and strong data type approaches to semantic integrity. Because of the preliminary nature of XSYS work, the following comments are provisional; based upon accrued experience, the basic XSYS design goals may be substantially modified in the future.

### 5.1 XSYS design

The major goal of XSYS is to provide a filter for checking remote database operations expressed via the XNDL. XSYS views semantic integrity maintenance as a system rather than user responsibility.

The offloading of XSYS from both target systems and their DBMSs permits a flexible, iterative approach to its design, implementation and modification. The design of XSYS is guided by several principles:

1. Offloading of semantic integrity enforcement—the burden of enforcing semantic integrity constraints on access by the network user should be offloaded, as much as possible, onto XSYS. This reflects the perception that local management may prove unwilling to incur an extra processing burden just to support remote users.
2. Emphasis on global checking—because of the belief that remote users are less knowledgeable users, and the desire to minimize local system overhead in supporting semantic integrity.
3. Modular design and implementation—to permit an iterative design, implementation, operation and modification cycle without impacting local systems or other XNDM components.

XSYS consists of two major components: One component supports constraint specification and maintenance. The constraint specification processor accepts a constraint specification and stores an intermediate representation in the semantic integrity tables which are linked to the XNDM data dictionary.

The second component supports constraint evaluation and enforcement. It is activated when a user issues a request in XNDL. Enforcement decomposes into those checks which can be performed independently of the target DBMSs, and those requiring retrieval of data from local databases. The former are primarily type checks while the latter test whether the appropriate assertions are satisfied.

The XSYS processing sequence consists of: i) receipt of a parsed tree representation of an XNDL statement by the XNDM translator, ii) performance of type constraint tests, iii) retrieval of dependent data required to process run-time

tests, iv) performance of run-time tests, and v) return of a condition code indicating whether a semantic integrity violation was detected and, if so, its type.

### 5.1.1 XSYS strong type constraints

XSYS supports strong data types whose specification includes:

1. Data name.
2. Data format—a description of the format of the data type as composed from primitive types such as character, integer, or real.
3. Legal operators—permissible for a given domain, e.g., arithmetic, relational, and DML operators which, for binary operators, will be domain dependent.
4. Compatible domain names—a list of domain names which can be involved in binary operations with the given domain.
5. Value restriction—assertions upon value such as legal range or permitted set of values.

This initial collection of capabilities is significantly less encompassing than those reported in [BRODM 78] but does provide a reasonable range of functionality. Moreover, as experience is gained into their utility, additional extensions can be implemented because of the modular nature of XSYS.

### 5.1.2 XSYS assertion processing

Key issues in supporting XSYS assertion processing are: i) maintenance of the appropriate database of assertions, ii) evaluation of the collection of assertions for consistency, and, iii) utilization of these assertions in performing the appropriate run time checks. The first two issues are currently being investigated. The third, given the nature of the networking environment, is affected by data movement requirements necessary to support run time processing. Such data movement can be reduced by having the local node perform some of the checking. This distributed approach has the obvious disadvantage of requiring local additions and modifications. In an operational environment, however, such modifications may prove highly cost effective.

XSYS assertions may involve interrelated data elements. Since a table based data model is being employed, such assertions can be classified into row, column, intra-table, or inter-table assertions. Evidently, processing of more complex assertions is likely to require retrieval of greater amounts of data from the supported DBMSs.

Consistency constraints can be further classified as: change rules, insertion rules, and deletion rules. Initially, only change rules are being implemented.

The distributed environment provided by XNDM imposes a bandwidth limitation on communications between the requesting process and the target DBMSs. As a result, it is highly desirable that such interactions have a transaction oriented flavor as observed in [GRAYJ 78]. In turn, this re-



quires capability for user specification of *when* semantic integrity rules are to be enforced. Since XSYS is external to the target DBMS, such temporary extensions are easy to effect. However, backing out an extension may prove rather difficult if the target system does not provide an appropriate collection of backout mechanisms.

### 5.1.3 XSYS system issues

XSYS semantic integrity specifications are maintained in tables associated with the XNDM data dictionary. Integrity maintenance commands are provided for DISPLAYing, DELETEing, DEFINEing and CHANGEing integrity specifications.

The cost of achieving a high degree of database semantic integrity may be prohibitive [BADAD 79; HAMMM 78]. The distributed nature of semantic integrity enforcement provided by XSYS naturally raises the issue of performance. Consequently, timing and frequency statistics are to be maintained with the integrity specifications for assessing invocation rates and performance penalties. Such information should permit estimating the cost and performance of assuring semantic integrity in the network database environment.

## 6. CONCLUDING REMARKS AND IMPLEMENTATION STATUS

The preceding discussion structured a basic approach to providing semantic integrity while supporting uniform access to multiple, remote, heterogeneous DBMSs. Moreover, the discussion has shown that the implementation framework of XSYS permits a blend of both strong data type and assertion based approaches to semantic integrity. Thus, a flexible vehicle supporting future research in this area has been achieved.

XSYS implementation is currently in a very preliminary state. Some strong data typing capabilities exist and the implementation of assertion checking has begun. Although the early implementation status precludes reporting operational results, the structure has been described to encourage discussion of this important issue.

## ACKNOWLEDGMENT

The authors would like to express their appreciation to L. J. Miller and Pearl S.-C. Wang for their helpful comments on earlier versions of this paper.

## REFERENCES

- [BADAD 79] Badal, D. Z., "On Efficient Monitoring of Database Assertion in Distributed Databases," *Proceedings of 4th Berkeley Conference on Distributed Data Management and Computer Networks*, August 1979, pp. 125-137.
- [BRODM 78] Brodie, Michael L., "Specification and Verification of Data Base Semantic Integrity," University of Toronto, Computer System Research Group, Technical Report CSRG-91, April 1978.
- [BUNEO 77] Buneman, O. Peter and Howard Lee Morgan, "Implementing Alerting Techniques in Database Systems," The Wharton School, University of Pennsylvania, DDC No. AD-A057319, February 1977.
- [CODAS 79] CODASYL Data Description Language Committee, "Data Description Language—Journal of Development, 1978," available from CODASYL, P.O. Box 1808, Washington DC, 20043, updated thru July 1979.
- [DAYAU 78] Dayal, Umeshwar and Philip A. Bernstein, "On the Updatibility of Relational Views," *Proceeding of 4th International Conference on Very Large Data Base*, West Berlin, Germany, Sept 1978, pp. 368-377.
- [ESWAK 75] Eswaran, Kapali P. and Donald D. Chamberlin, "Functional Specifications of a Subsystem for Data Base Integrity," *Proceeding of 2nd International Conference on Very Large Data Base*, Framingham, MA., Sept 1975, pp. 48-68.
- [FERNE 76] Fernandez, Eduardo B. and Rita C. Summers, "Integrity Aspects of a Shared Data Base," *AFIPS Conference Proceeding, National Computer Conference*, Vol. 45, 1976.
- [GRAVR 75] Graves, Robert W., "Integrity Control in a Relational Data Description Language," *Proceedings of ACM Pacific Regional Conference*, 1975, pp. 108-113.
- [GRAYJ 78] Gray, James, "Notes on Data Base Operating Systems," in Goos and Harmanis (eds.), *Operating Systems—An Advanced Course*, Lecture Notes in Computer Sciences, Vol. 60, pp. 393-481.
- [GUTTJ 77] Gutttag, J. V., "Abstract Data Types and the Development of Data Structures," *Communications ACM* 20, 6 (June 1977), pp. 396-404.
- [HAMMM 75] Hammer, Michael and Dennis J. McLeod, "Semantic Integrity in a Relational Data Base System," *Proceeding of 2nd International Conference on Very Large Data Base*, Framingham, MA., Sept 1975, pp. 25-47.
- [HAMMM 76] Hammer, Michael and Dennis J. McLeod, "A Framework for Data Base Semantic Integrity," in *Proceedings of 2nd International Conference on Software Engineering*, San Francisco, CA, Oct. 1976.
- [HAMMM 78] Hammer, Michael and Sunil K. Sarlin, "Efficient Monitoring of Data Base Assertions," *Proceeding of ACM SIGMOD 1978*, p. 159.
- [KARRM 78] Karr, Michael and David B. Loveman III, "Incorporation of Units into Programming Languages," *Communications of ACM* Vol. 21, No. 5 (May 1978), pp. 385-391.
- [KIMBS 79] Kimbleton, S. R., Pearl Wang and Elizabeth Fong, "XNDM: An Experimental Network Data Manager," *Proceedings of the 4th Berkeley Workshop on Distributed Database Management*, August 1979, pp. 3-17.
- [LISKB 74] Liskov, B. H. and S. N. Zilles, "Programming with Abstract Data Types," *Proceedings of ACM SIGPLAN Symposium on Very High Level Language*, SIGPLAN Notices 9,4 (April 1974), pp. 50-59.
- [MACHC 76] Machgeels, Claude, "A Procedural Language for Expressing Integrity Constraints in the Coexistence Model," in Nijssen, G. M. (ed.) *Modelling in Data Base Management Systems*, North Holland Publishing Company, 1976.
- [MCLED 76] McLeod, Dennis J., "High Level Expression of Semantic Integrity Specifications in a Relational Data Base System," MIT Report MIT/LCS/TR-165, available from DDC AD-A034184, 1976.
- [MELOR 79] Melo, Rubens N., "Monitoring Integrity Constraints in a CODASYL-like DBMS," *Proceedings of 5th International Conference on Very Large Database*, Rio de Janeiro, Brazil, Oct. 1979, pp. 209-218.
- [STONM 75] Stonebraker, M., "Implementation of Integrity Constraints and Views by Query Modification," *Proceedings of ACM SIGMOD International Conference on the Management of Data*, San Jose, California, May 1985.
- [WANGP 80] Wang, P. S. C., "The Design of a Network Virtual Query Language-XNQL" (in preparation).
- [WEBEH 76] Weber, Herbert, "A Semantic Model of Integrity Constraints on a Relational Data Base," in Nijssen, G. M. (ed.) *Modelling in Data Base Management Systems*, North Holland Publishing Company, 1976.
- [WOODH 79] Wood, Helen M. and Stephen R. Kimbleton, "Access Control Mechanisms for a Network Operating System," *Proceeding of National Computer Conference*, June 1979.

# Concurrency coordination in a locally distributed database system\*

by GRUIA-CATALIN ROMAN

Department of Computer Science  
Washington University  
St. Louis, Missouri

## INTRODUCTION

A database is called internally consistent (or just consistent) with respect to a given set of invariant properties if, in the absence of any database activity, the above said invariants hold true.<sup>1</sup> These properties reflect relations among objects of the application domain and characterize its semantic consistency (e.g., 'last name appears first on all government forms'). Because a single database may support many different applications, these relations are generally assumed to be known solely by the database user and not by the database designer. The very definition of a transaction (a sequence of primitive database activities which, when acting alone, preserves, as a whole, all invariants) is in recognition of the fact that it is the user's duty to guarantee the integrity of the data he manipulates.

While the assumption that during the execution of one transaction no other transaction exists in the system is reasonable from the user viewpoint, it is not acceptable from a design perspective since it would preclude any concurrency. It is the designer's task to assure the preservation of the user viewpoint while maximizing performance through the use of concurrency. The problem of developing concurrency control or access synchronization algorithms is perhaps the most difficult issue facing the database designer today. On one hand, while each transaction by itself preserves database consistency, concurrent execution of several transactions may ultimately result in: (1) the violation of the invariants persisting even after termination of all involved transactions and (2) the reading of inconsistent data by some of the transactions. On the other hand, excessive overhead in propagating or waiting for concurrency control messages can seriously degrade database performance. Starting in the early seventies and more extensively during the last three years, concurrency coordination has received considerable attention among database research groups. The very first approach to be considered for the synchronization of concurrent updates in distributed databases was the use of locks (shared and exclusive).<sup>2,3</sup> However, locking, which

was suitable in centralized databases, proved to be very inadequate for geographically distributed databases due to extreme communication delays.

Consequently, efforts have been made toward the development of more effective coordination schemes which reduce the number of messages required to propagate locking information, e.g., Thomas<sup>4</sup> and Ellis.<sup>5</sup> The savings gained by these approaches are, however, insufficient when a large number of sites and high transaction volume are involved. Other authors attempted to bring forth improvements through the incorporation of a minimal level of centralization with a distributed system, e.g., Alsberg and Day,<sup>6</sup> Stonebraker and Neuhold.<sup>7</sup> The performance of such algorithms depends upon the degree to which a proper partitioning of database activities is achievable.

Still another strategy was adopted by Bernstein, et al.<sup>8</sup> This method takes advantage of an a priori classification of transactions based upon the different nature of their interactions. In this manner locking efficiency is achieved by avoiding unnecessary global locking when possible.

A "best" algorithm has not yet been found, and the search continues. This paper is concerned with highly distributed local databases anticipated to emerge due to the advent of VLSI technology. A pipelined architecture for a multi-processor database system is proposed along with a concurrency coordination scheme. The architecture can be modeled as a collection of single-rooted directed acyclic graphs (DAG's). The root of each DAG is associated with a column of a cross-bar switch; the rows correspond to separate user groups. Each node of the DAG represents a processor, and each arc indicates a communication link between processors. The leaves of the DAG's are called *data processors* and may contain single files or larger portions of the database. The other nodes, called *directory processors* are assumed to store, in a distributed fashion, an index structure designed to maximize throughput while maintaining a relatively constant response time. For the sake of simplifying the exposition, changes to the index structure will not be considered here.

Transactions are entered serially at the root of each DAG and the concurrency coordination mechanism enforces their "proper" pipelining through the networks. Each directory

\* This work has been supported in part by the Division of Research Resources of the National Institutes of Health under Grant RR 0396.

processor performs a routing function. When the leaves are reached, the requested data is read and passed back to the user who later sends the updates followed by a commit message. The sending of a commit message is fundamental to the approach; the fact that the user creates the updates is not. An optimized system would have the data and/or directory processors create the updates. The decision would be determined by some data transfer volume minimization strategy.

The remainder of this paper includes a short review of the terminology, a formal description of the proposed scheme, and a discussion of several performance related issues.

## DEFINITION OF TERMS

The terms "database," "transaction" and "consistency" are used as defined by Gray.<sup>9</sup> Each transaction is assumed to have three phases:

- (1) Data Selection Phase—A selection criterion or query is passed to the root nodes of the networks (DAG's). Each root node distributes the selection process among its successors by initiating appropriate subqueries. Later, when the answers to the subqueries return, it concatenates and sends them to the user along with an end-of-selection signal. All other nodes repeat the same scenario throughout the network.
- (2) Data Modification Phase—Upon receipt of the required data, the necessary updating is carried out.
- (3) Data Commitment Phase—The updated data is sent back into the network to permanently replace the old copies. As soon as all updates are acknowledged, a final commit message is issued by the user.

Some notation is introduced next as a necessary tool for enabling a formal treatment of the special case where a single DAG is present. The extension to multiple networks is made, in an informal manner, later on.

- $X_z$  denotes a single node (data or directory processor) in the network.  $X_O$  is used to represent the root node.
- $T_i$  represents the transaction  $i$ . Each transaction is assumed to have a unique identifier  $i$  (e.g., distinct time stamps). All messages exchanged by processors in behalf of  $T_i$  carry the identifier  $i$ .
- $Q_i$  denotes any non-null message associated with the transaction  $T_i$  during the data selection phase; it is called a *query message*. The notation  $Q_i \in T_j$  means  $i=j$ .
- $i$  is used to represent a null query message associated with  $T_i$ .
- $S_k$  is called a schedule and is defined as an arbitrary sequence of query messages.  $S_k$  is a *serial schedule* iff no two query messages in  $S_k$  have the same identifier, i.e.,  
 $(S_k)_{p \in T_i} \& (S_k)_{q \in T_j} \& p \neq q \Rightarrow i \neq j$   
 where  $(S_k)_r$  is the  $r^{\text{th}}$  query message in  $S_k$ .
- A period denotes schedule concatenation.

## CONCURRENCY COORDINATION

It has already been stated that concurrent execution of several transactions can result in violations of database consistency due to undesirable interferences among transactions. The network organization suggested in this paper adds a new level of complexity since each transaction is present concurrently in many nodes of the DAG. Thus, concurrency coordination is required even when a single transaction exists in the system. The coordination scheme described below handles uniformly both types of situations. Furthermore, it assumes no centralization, is simple, and involves a small synchronization overhead. However, in contrast to previously referenced approaches, it is architecture specific.

The idea behind the scheme is the following. Given any two data processors,  $X_p$  and  $X_q$ , for any two transactions  $T_i$  and  $T_j$  which access (read or write) data from both  $X_p$  and  $X_q$ , the two processors are forced to see  $T_i$  and  $T_j$  in the same order. This can be accomplished by requiring the behavior of every node to be describable by the functions below:

( $S$  is the set of all possible serial schedules)

MATCH:  $S^n \rightarrow S$   
 $SO = \text{MATCH}(S_1, S_2, \dots, S_n)$   
 where  
 $(SO)_{l \in T_i} \quad \text{iff} \quad (S_j)_{l \in T_i} \quad \text{for } j=1, 2, \dots, n$   
 $(SO)_{l=} \begin{cases} Q_i & \text{iff } \exists k: (S_k)_{l=} Q_i \\ i & \text{iff } (S_j)_{l=} i \quad \text{for } j=1, 2, \dots, n \end{cases}$

TRIM:  $S^n \times S \rightarrow S^n$   
 $(S_1', S_2', \dots, S_n') = \text{TRIM}(S_1, S_2, \dots, S_n; SO)$   
 where for  $j=1, 2, \dots, n$   
 $S_j = S_j'' \cdot S_j'$   
 $(S_j'')_{l \in T_i} \text{ iff } (SO)_{l \in T_i}$

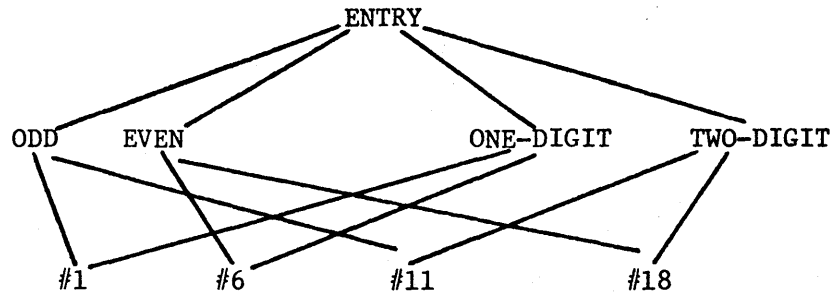
FORK:  $S \rightarrow S^m$   
 $(S_1, S_2, \dots, S_m) = \text{FORK}(SO)$   
 where for  $j=1, 2, \dots, m$

$$(S_j)_{l=} \begin{cases} Q_i \text{ or } i & \text{if } (SO)_{l \in T_i} \& (SO)_{l=} i \\ i & \text{if } (SO)_{l=} i \end{cases}$$

Each node  $X_u$  processes some transaction  $T_i$  if and only if it receives a message associated with  $T_i$  from all its predecessors in the DAG. Some of the messages may be null, some may represent update requests, and others may be read requests.  $X_u$  (conceptually) combines all messages into a single one. The MATCH function simulates precisely this action while TRIM is used to express the removal of the respective messages from all input queues. It is essential for  $X_u$  to wait until all messages associated with  $T_i$  are received; otherwise, improper transaction processing and destruction of the serialization would occur.

The interpretation of the combined query, built on behalf of the transaction  $T_i$ , results in subqueries (associated with  $T_i$ ) to be sent to some of the successors of  $X_u$ . However, in order to assure correct propagation of the schedule, all

DATABASE CONFIGURATION



PROCESSING OF THE SCHEDULE T1,T2

ENTRY:	(T1 : add 2 mod 6 to one-digit numbers).(T2 : list even numbers)	Q1.Q2
ODD:	(T1 : nil).(T2 : nil)	1.1
EVEN:	(T1: nil).(T2 : list even numbers)	1.Q2
ONE-DIGIT:	(T1 : add 2 mod 6 to one-digit numbers).(T2 : nil)	Q1.2
TWO-DIGIT:	(T1 : nil).(T2 : nil)	1.2
#1:	(T1 : nil).(T2 : nil) = (T1 : add 2 mod 6).(T2 : nil)	Q1.2
#6:	(T1 : nil).(T2 : list) = (T1 : add 2 mod 6).(T2 : list)	Q1.Q2
#11:	(T1 : nil).(T2 : nil) = (T1 : nil).(T2 : nil)	1.2
#18:	(T1 : nil).(T2 : list) = (T1 : nil).(T2 : list)	1.Q2

Figure 1—Sample schedule processing

other successors of  $Xu$  also receive messages on behalf of  $Ti$ . These messages are null messages needed solely for concurrency coordination purposes. The function FORK is non-deterministic and simulates the process by which  $Xu$  decides what query messages to send to its successors as a consequence of processing a combined query message generated by MATCH. Since each node processes and puts out messages in the same order, the initial total order over the transactions  $Ti$  is maintained throughout the network.

### Proposition

Given the fact that the entry node  $XO$  is presented with a serial schedule  $SO$ , every node  $Xu$  will execute an order equivalent serial schedule  $Su$ :

$$(Su)/\leq Ti \text{ iff } (SO)/\leq Ti.$$

### Proof by induction

- (O) The proposition holds trivially for  $XO$   
 MATCH(SO) = SO  
 TRIM(SO; SO) =  $\emptyset$   
 FORK(SO) = (SO1, SO2, ..., SO $m$ )—where SO $k$  represents the schedule generated by SO for its  $k^{\text{th}}$  successor and is order equivalent to SO.
- (N) Let us assume the proposition to be true for nodes at distance  $N$  from  $XO$ . Distance is defined as the length (number of links) of the longest path from  $XO$  to the particular node.
- (N+1) Given any node  $Xu$  at distance  $N+1$ , due to assumption (N) all its input schedules are order equivalent to SO. By applying again the definitions of MATCH, TRIM, and FORK, one establishes the propositions to be true for the level  $N+1$ .

Figure 1 describes a database and the sample schedules seen by each node as a result of processing two transactions,  $T1$  and  $T2$ . The reader should note, however, that in the example of Figure 1, the updates are carried out at the nodes. If node #6, for instance sends data to the entry point to be updated by the user processor that initiated the particular transaction, node #6 will not process the next message until the result comes back and is committed throughout the network.

The extension of this coordination scheme from the one to several networks requires that all root nodes receive transactions spanning across nets in the same order. This can be achieved through the use of a hardware cross-bar switch, Figure 2. When a user terminal issues a transaction which concerns only one of the database networks, the query message transmission takes place as soon as the vertical path becomes available. However, if the query involves more than one network, the user must request allocation of all needed paths before sending query messages to the various networks. This strategy will guarantee that the separate serial schedules are all consistent with each other, i.e., any

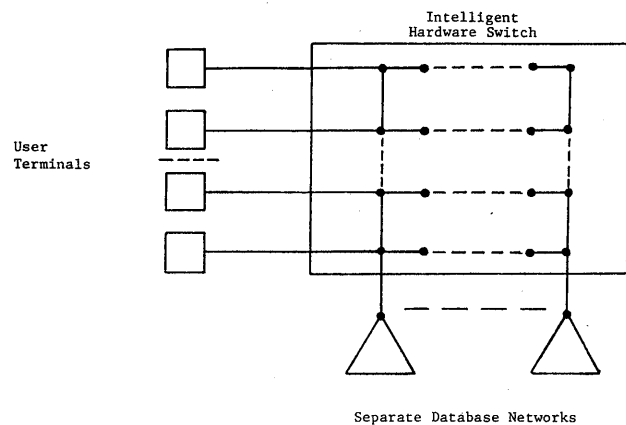


Figure 2—Use of a Hardware Cross-Bar Switch.

two transactions occur in the same order in any serial schedule in which they appear together. Furthermore, if switching paths are viewed as resources and always allocated in the same order, the possibility of deadlock is eliminated. At the same time, blocking within the switch can be minimized by allowing single network users to use paths which have been allocated to a multiple network user but are not yet in use.

### PERFORMANCE ISSUES

The architectural solution described above has its justification in the application domain for which it is intended—medical information systems. Such systems tend to be confined to a single geographical location, grow relatively fast, require quick response, and exhibit a processing pattern dominated by data retrieval and creation rather than updates. As such, modifications of the directories, other than additions of new entries, can be assumed to be few. Therefore, the user should be willing, in those rare occasions, to pay an additional waiting penalty for coordinating the concurrent update of several directories.

The distribution of the index structure over several directory processors is meant to reduce the searching time through the use of concurrency in a pipelined-like fashion. The goal is to assure a good average time response through the addition of new directory and data processors when faced with transaction volume increases. However, the system's ability to handle the higher throughput relates not only to the number of processors being used but also to the "appropriateness" of the data and index distribution. Ideally, all data processors should be equally utilized. Furthermore, the searching load within each net should be equally distributed among directory processors at equal distance from the root since the coordination scheme forces each processor to work at the rate of the slowest predecessor.

With respect to transaction recovery and roll-back, a transaction failure in some node could be signaled by passing a failure message to the user processor, which, in turn would

send a "forget about my updates" message in place of the commit. Subsequently, the transactions would be started again with a new identifier. A node failure would have the effect of cancelling any transaction that requires its use.

#### SUMMARY

An architecture for a locally distributed database system was suggested. A simple solution to the problem of coordinating concurrent transactions within the database was presented. The solution requires no centralized control, is deadlock free, uses no locks, is fair, and involves little overhead.

#### REFERENCES

1. Eswaran, K. P., Gray, J. N., Lorie, R. A., and Traiger, I. L., "The Notions of Consistency and Predicate Locks in a Database System," *Communications of ACM*, Vol. 19, No. 11, 624-633, November 1976.
2. Rosenkrantz, D. J., Stearns, R. E., and Lewis, P. M., "System Level Concurrency Control for Distributed Database Systems," *ACM Transactions on DB Systems*, Vol. 3, No. 2, 178-198, June 1978.
3. Stucki, M. J., Cox, Jr., J. R., Roman, G.-C., and Turcu, P. N., "Coordinating Concurrent Access in a Distributed Database Architecture," *Proceedings Fourth Workshop on Computer Architecture for Non-Numeric Processing*, Syracuse University, August 1978.
4. Thomas, R. H. and Henderson, D. A., "McRoss—A Multi-Computer Programming System," Spring Joint Computer Conference, 281-293, 1972.
5. Ellis, C. A., "A Robust Algorithm for Updating Duplicate Databases," *Proceedings 2nd Berkeley Workshop on Distributed Data Management and Computer Networks*, 146-158, May 1977.
6. Alsberg, P. A. and Day, J. D., "A Principle for Resilient Sharing of Distributed Resources," Center for Advanced Computation, University of Illinois, Urbana, Illinois, 1976.
7. Stonebraker, M. and Neuhold, E., "A Distributed Data Base Version of INGRES," Memo No. ERL-M612, 11, September (1976) University of California, Berkeley, California.
8. Bernstein, P. A., Rothnie, J. B., et al., "The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (The Fully Redundant Case)," *IEEE Transactions on Software Engineering*, Vol. SE-4, No. 3, May 1978.
9. Gray, J. N., "Notes on Data Base Operating Systems," Research Report RJ2188(30001) IBM Research Laboratory, San Jose, California 95193, 1978.



# An introduction to computed chaining

by KUO-CHUNG TAI and ALAN L. THARP

North Carolina State University  
Raleigh, North Carolina

## INTRODUCTION

A hashing function,  $H(x)$ , is a transformation from a key value  $x$  to an address. Since such transformations may produce the same address for distinct keys, hashing functions lead to collisions in the address space. Many methods for resolving hashing collisions have been reported [1,2]. Direct chaining and open addressing are the two basic collision-resolution methods. This paper presents a form of hybrid hashing, *computed chaining*, which is better than a previously described hybrid hashing procedure called pseudochaining [3]. In certain situations it is also better than other collision procedures.

In direct chaining, distinct items which hash into the same home address are linked into a chain. When a collision occurs while inserting an item  $x$ , if the item at  $H(x)$  is stored at its home address, then  $x$  is stored at the first empty cell encountered on the chain starting at  $H(x)$ . If the item at  $H(x)$  is not stored at its home address, then that item is moved into an empty cell, its chain is updated, and  $x$  is inserted at  $H(x)$ . The links in direct chaining are addresses of overflow items. Figure 1(a) shows the use of direct chaining for items  $A$ ,  $B$ , and  $C$  which have the same home address,  $r$ , and are inserted in the given order. Only one probe is required to retrieve an overflow item from its predecessor.

In open addressing, when a collision occurs while inserting an item  $x$ , a function of  $x$  is used to determine the probe sequence. The first empty cell encountered in the probe sequence is used to store item  $x$ . The same probe sequence will later be used to retrieve item  $x$ . Many probe-sequence generating functions have been studied [1,2]. Figure 1(b) illustrates the use of open addressing for items  $A$ ,  $B$ , and  $C$ . The number of probes required to locate an empty cell for storing an overflow item is a function  $f$  of that item and all other items in the table, and that same number of probes is later required to retrieve that item.

Pseudochaining [3] combines features of direct chaining and open addressing in the following way. Suppose that item  $x$  is stored at its home address  $H(x)$ . When the first item  $y$  with  $H(y)=H(x)$  ( $y$  is called the first overflow item) is inserted, an empty cell for storing  $y$  is located using open addressing and the number of probes required to locate the empty cell is stored in the link field of  $H(y)$ . By doing so, only one probe is required to retrieve  $y$  from its home address

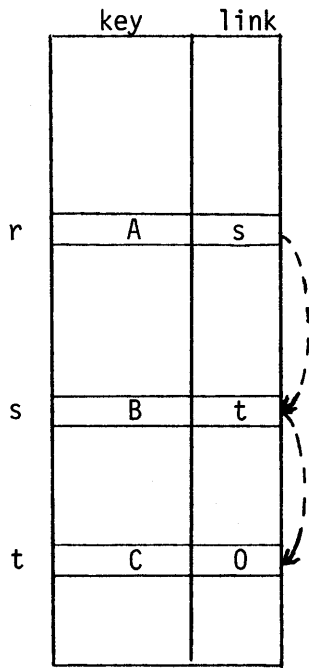
because the address of  $y$  can be computed from the probe number stored in the link field of  $H(y)$ . Since the probe number typically requires fewer bits than the full address, pseudochaining needs less link space than direct chaining. Subsequent overflow items with a home address of  $H(y)$  are also stored using open addressing, but they do not necessarily have the property of one-probe retrieval from  $H(y)$ . Figure 1(c) shows the use of pseudochaining for items  $A$ ,  $B$ , and  $C$ .  $f(B)$  is stored as the link of  $A$  so that only one probe is required to retrieve  $B$  from the address of  $A$ .  $f'$  is the modified  $f$  after  $B$  is inserted (see [3] for details).

Computed chaining is another form of hybrid hashing that uses probe numbers as links instead of actual addresses. This new method, however, is closer to direct chaining than is pseudochaining, because all items with the same home address are linked into a chain. Assume item  $x$  is stored at its home address,  $H(x)$ . When inserting the first overflow item  $y$  with  $H(y)=H(x)$ , computed chaining uses a function of  $x$  (not  $y$ ) to determine the probe sequence and then stores (in the link field of  $H(x)$ ) the number of probes required to find an empty cell for storing  $y$ . Later when inserting the second overflow item  $z$  with  $H(z)=H(x)$ , computed chaining first computes the address of  $y$  using item  $x$  and its link. Then this method uses a function of  $y$  to determine the probe sequence starting from the address of  $y$ . It stores as the link of  $y$  the number of probes required to find an empty cell for storing  $z$ . By doing so, an item plus its address and link (a probe number) determine the address of the next item in the same chain. Thus, items with the same home address are linked into a chain without the addresses being stored; instead the addresses can be *computed* by using the probe numbers stored in the link fields. This method is similar to direct chaining in that only one probe is required to retrieve an overflow item from its predecessor. Figure 1(d) illustrates the use of computed chaining for items  $A$ ,  $B$ , and  $C$ . The number of probes required to locate an empty cell for storing an overflow item is a function  $g$  of its predecessor. By storing  $g(A)$  ( $g(B)$ ) as the link of  $A$  ( $B$ ), only one probe is required to retrieve  $B$  ( $C$ ) from the address of  $A$  ( $B$ ).

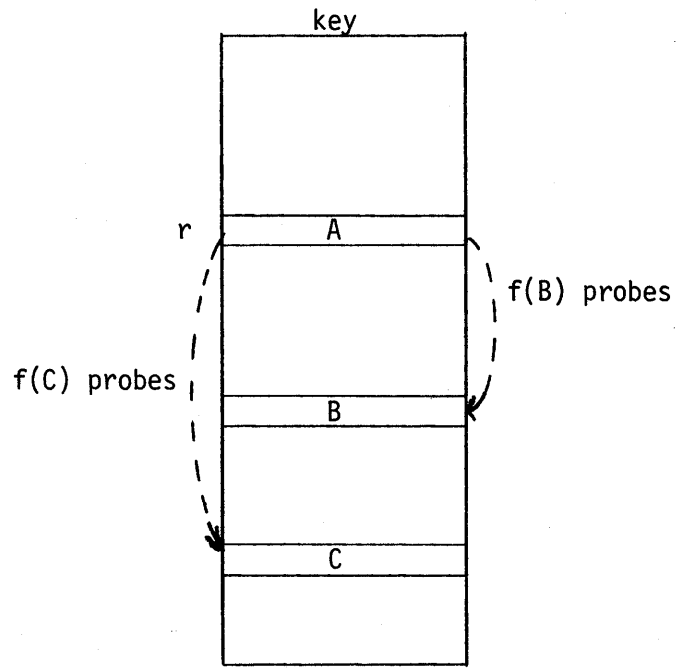
## DESCRIPTION OF COMPUTED CHAINING

The previous section highlighted computed chaining and compared and contrasted it with other well-known hashing

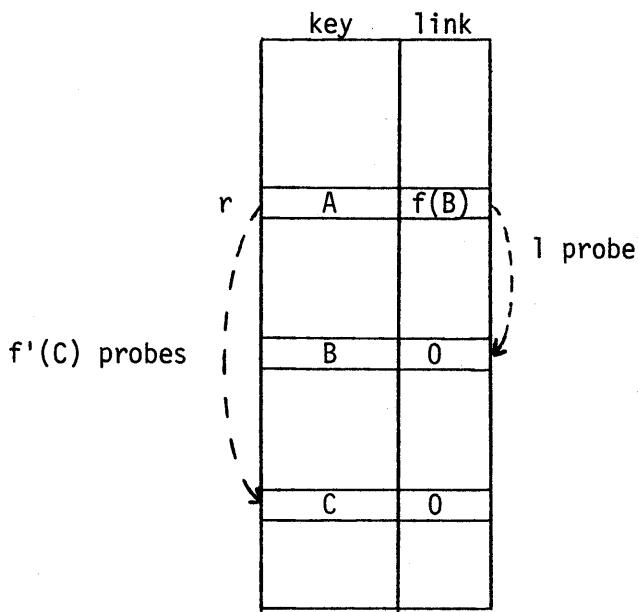




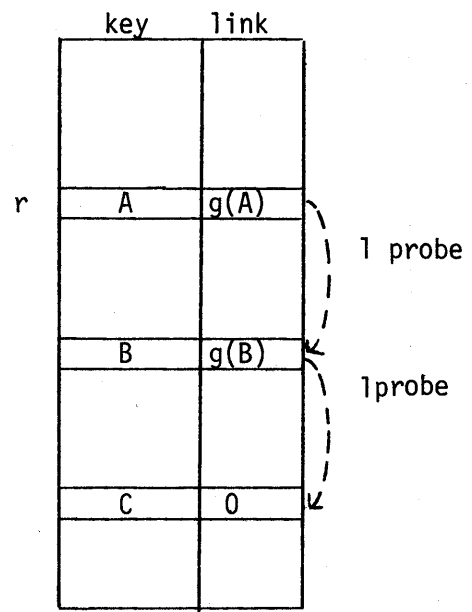
(a) direct chaining



(b) open addressing



(c) pseudo-chaining



(d) computed chaining

Figure 1—Illustration of the basic hashing mechanisms.

methods. A more detailed description of computed chaining will be given for the operations *probe*, *store*, *move* and *retrieve*. *Probe* computes the next probe address based upon the linear quotient hashing method. *Store* places an item into

the table according to the computed chaining method. *Move* displaces an item not stored at its home address and all subsequent items stored on that chain. And finally, *retrieve* locates an item in the table. For this discussion, the linear

quotient hashing scheme [4] is used as the hashing function and method for determining the address of the next probe. Computed chaining is not dependent upon the linear quotient hashing scheme; hashing functions other than the one which follows may be used with computed chaining.

procedure probe (*k*, prob#, addr)

//computes the next probe address by the linear quotient method using the key, probe number and current address//

1.  $\text{incr} \leftarrow (k/p) \bmod p$  //obtain increment//
2. if  $\text{incr} = 0$  then  $\text{incr} \leftarrow 1$  //insure a positive increment//
3. return  $((\text{addr} + \text{prob\#} * \text{incr}) \bmod p)$  //return new address//

end probe

*k* is the key, prob# is the probe number and addr is the current address in the table. *p* is the prime number table size.

Using the *probe* procedure, *store* locates the first empty cell or one that contains an item not at its home address. In the latter case, a call is made to the *move* procedure to free the cell for insertion. The new item is then inserted. If the new item is an overflow item, the number of probes for locating the new item from its predecessor is stored in the link field of the predecessor. To subsequently access the new item, only a "single" probe from the predecessor is needed because the address of the new item can be computed from the address, key and link value of the predecessor.

procedure store(*x*)

//stores an item with key *x* according to the computed chaining hashing method//

1.  $h \leftarrow x \bmod p$  //locate the home address using the linear quotient method//
2. if  $T.h = 0$  then [ $T.h \leftarrow x$ ; return] //the home address is empty, so store the item//
3. If  $T.h = x$  then return //item is a duplicate//
4. if  $T.h \bmod p \neq h$  then  
[call move (*h*);  $T.h \leftarrow x$ ; return] //item stored at *h* is not at its home address, so move it and store *x* at *h*//  
//locate the last item in this chain//
5. while  $L.h \neq 0$  do
6.  $h \leftarrow \text{probe}(T.h, L.h, h)$  //locate the next probe address using the probe number stored at *h*, *L.h*//
7. if  $T.h = x$  then return //item is a duplicate//
8. end
9.  $i \leftarrow 1$  //initialize a loop variable to locate the first empty cell for storing *x*//
10.  $j \leftarrow \text{probe}(T.h, i, h)$  //locate the next probe address//
11. while  $T.j \neq 0$  do
12.  $i \leftarrow i + 1$
13. if  $i > p$  then [print('overflow'); stop]
14.  $j \leftarrow \text{probe}(T.h, i, h)$  //locate the next probe address//

15. end

16.  $L.h \leftarrow i$  //insert the probe number//

17.  $T.j \leftarrow x$  //store the item//

end store

*T.h* refers to the key at location *h* and *L.h* is the link value at location *h*.  $T.h = 0$  means that the cell at location *h* is empty. Initially  $T.h = 0$  and  $L.h = 0$  for  $0 \leq h \leq p - 1$ .

The *move* procedure in computed chaining is essential in minimizing the number of retrieval probes. As mentioned previously, it moves an item not at its home address. Since the address and link of the item being moved is used to locate subsequent items in the same chain, it is necessary then to move all of these subsequent items. A simple solution is to use an array of temporary storage locations to queue all items to be moved. Then one by one (first-in-first-out) the items stored in the array are moved to new locations. By doing so, the order of the moved items remains the same after moving. (It is unnecessary to keep the same order or to use the array during moving. A better solution which does not need the array (or stack) exists, but it is not described here due to its complexity.) In this moving operation, the associated probe numbers are updated. Finding the new locations of the moved items is essentially the same as in the *store* operation. Note that the *move* operation and the special care associated with its use are performed *only* during the insertion of an item. If coalescence of multiple chains is allowed, then the *move* procedure is unnecessary but the performance of computed chaining will be degraded somewhat (see [1, pp. 514-518]). The *move* operation is recommended when the ratio of insertions to retrievals is quite small.

procedure move(*r*)

//moves an item at location *r* not stored at its home address and all subsequent items in that computed chain//

1. declare TEMP(1:100) //declare array for storing *r* and subsequent items in the computed chain//
2.  $h \leftarrow T.r \bmod p$  //locate home address for item stored at *r*//  
//locate the item preceding that at *r* in the computed chain//
3. while  $\text{probe}(T.h, L.h, h) \neq r$  do
4.  $h \leftarrow \text{probe}(T.h, L.h, h)$
5. end
6.  $i \leftarrow 1$  //initialize array index//
7.  $\text{TEMP}(i) \leftarrow T.r$  //store the item at *r*//  
//find subsequent items and store them in TEMP//
8.  $y \leftarrow r$
9. while  $L.y \neq 0$  do
10.  $\text{nexty} \leftarrow \text{probe}(T.y, L.y, y)$  //find the next item//
11.  $i \leftarrow i + 1$  //increment the array index//
12.  $\text{TEMP}(i) \leftarrow T.\text{nexty}$  //store the item found//
13.  $T.y \leftarrow 0$  //erase the item at *y*//
14.  $L.y \leftarrow 0$
15.  $y \leftarrow \text{nexty}$  //continue the search//
16. end
17.  $T.y \leftarrow 0$  //erase the last item//

```

//restore items in TEMP on the first-in-first-out
//basis so that the order of items in the computed
//chain remains the same//
18.  $ii \leftarrow 1$  //initialize an array index for moving items//
19. while  $ii = < i$  do
20.    $k \leftarrow 1$  //initialize a loop variable to locate the
//first empty cell for storing the item in
//TEMP(ii)//
21.    $j \leftarrow \text{probe}(T.h, k, h)$  //locate the next probe
//address//
22.   while  $(T.j \neq 0$  or  $j = r)$  do
23.      $k \leftarrow k + 1$ 
24.     if  $k > p$  then print ('overflow') //table is full//
25.      $j \leftarrow \text{probe}(T.h, k, h)$  //locate the next probe
//address//
26.   end
27.    $L.h \leftarrow k$  //store the probe number//
28.    $T.j \leftarrow \text{TEMP}(ii)$ 
29.    $ii \leftarrow ii + 1$  //prepare to move the next item in
//the chain//
30.    $h \leftarrow j$ 
31. end
end move

```

The final procedure, *retrieve*, is similar to the linear quotient retrieval scheme except for the probe function used to calculate the next probe address.

```

procedure retrieve(x)
//retrieves an item with key x which was stored
//according to the computed chaining hashing method//
1.  $h \leftarrow x \bmod p$  //locate the home address using the
//linear quotient method//
2. if  $T.h = x$  then return
3.  $i \leftarrow 1$  //initialize a loop variable for searching x//
4. while  $(i = < p$  and  $L.h \neq 0)$  do
5.    $h \leftarrow \text{probe}(T.h, L.h, h)$  //locate the next probe
//address//
6.   if  $T.h = x$  then return
7.    $i \leftarrow i + 1$  //increment the loop variable//
8. end
9. print ('item is not in the table')
end retrieve

```

If space is at a premium and sufficient bits in the link field are not available to store the "complete" probe number, a technique from pseudochaining of using a function of the "complete" probe number (*prob#*) and the number of bits of the link field (*s*) can be incorporated into computed chaining. This function,  $\text{GBD}(\text{prob}\#, s)$ , gives the greatest divisor of *prob#* that is less than  $2^{**}s$  and relatively prime to *p*. To incorporate this modification into the previously defined procedures, it is necessary to replace statement 16 in *store* with

```
 $L.h \leftarrow \text{GBD}(i, s)$  //insert the probe number//
```

and line 27 in *move* should be replaced with

```
 $L.h \leftarrow \text{GBD}(k, s)$  //store the probe number//
```

In addition, the use of the *probe* procedure to both find a successor and insert an item should be changed.

First, consider the case of using the *probe* procedure for finding a successor. Assume that while inserting item *y*, the number of probes required to locate an empty cell for *y* from its predecessor *x* is *i*. Then  $\text{GBD}(i, s)$  is stored in the link field of *x*. To retrieve *y* from *x*, when the link field has sufficient bits ( $i < 2^{**}s$ ), only one probe is necessary. However, if insufficient bits ( $i \geq 2^{**}s$ ) are available, multiple probes may be required (see Figure 2). Let item *w* be one of the intermediate cells probed. Then one of these cases must hold:

- (1) *w* is not an item in the chain. *w* could be zero as the result of a move or deletion.
- (2) *w* is an item in the chain, but it precedes *x*, or
- (3) *w* is an item in the chain, but it follows *y*.

Note that case (3) should not be allowed, for if it were, items between *x* and *w* would not be accessible. To avoid case (3), care must be taken when inserting an item using the *probe* procedure as described in the next paragraph. If case (3) does not occur, then cases (1) and (2) can be detected easily because *w* is in the chain if and only if  $w \neq 0$  and either  $H(w) = H(x)$  if no coalescence is allowed, or  $H(w) = H(z)$  where *z* is any of the items preceding *w* in the chain if coalescence is allowed.

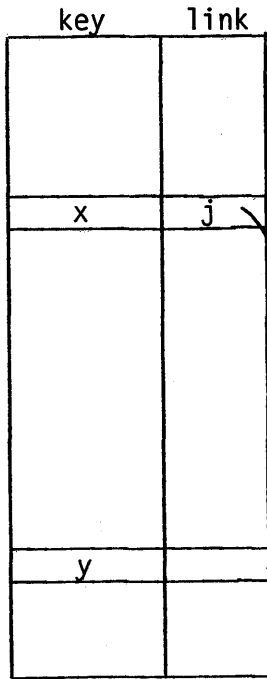
Then consider the case of using the *probe* procedure to find an empty cell for inserting an item. To avoid case (3) in subsequent retrieval, the empty cell must not be any of the empty cells which have been encountered prior to finding the last item of the chain on which the item is being inserted.

Two methods for both locating a successor and inserting an item when using the GBD function are suggested. One method would be to add a bit field to each cell in storage. This bit would signify whether a cell had been encountered previously in the same *store*, *retrieve* or *move* operation. When locating a cell with the *probe* procedure, this extra bit would be checked. Another procedure to accomplish the same result would be to establish a stack for storing, for subsequent interrogation, cells (or their addresses) previously encountered in the *probe* procedure. For a table of relatively short items, the latter method would be more space efficient.

The computed chaining technique described above is a hybrid of direct chaining and the linear quotient method of open addressing. Other open addressing methods can also be used in computed chaining. The reason that linear quotient was chosen is that it has the best performance among all "static" hashing methods (i.e., those which do not reorganize the table when inserting new items).

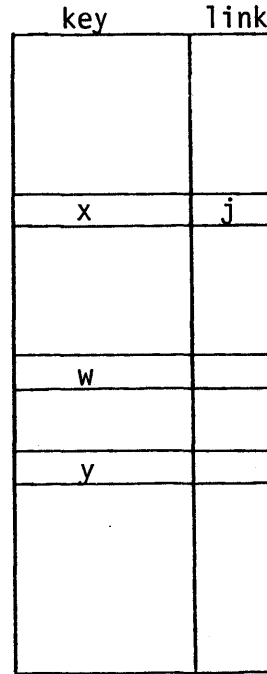
The idea of reorganizing the table when inserting new items, which is used in "dynamic" hashing methods [5-10], can be used in computed chaining to improve its perform-

$$j = \text{GBD}(i, s)$$



one probe

(a)  $i < 2 \cdot s$   
( $j = i$ )



(i/j) probes

(b)  $i \geq 2 \cdot s$   
( $j < i$  and is a divisor of  $i$ )

Figure 2—Locating successor item with computed chaining.

ance. It is believed, however, that the improvement due to dynamic hashing is only marginal.

Linear probing, a static hashing method, can be incorporated into computed chaining with less difficulty than linear quotient. The probe sequence in linear probing for an item  $x$  is

$$H(x), H(x) + b, H(x) + 2 \cdot b, \dots, H(x) + i \cdot b, \dots$$

where  $b$  is a constant. In computed chaining,  $b$  can be defined as a function of the item stored at  $H(x)$  and thus all items with the same home address will use the same  $b$ . Figure 3(a) illustrates this scheme in contrast to Figure 1(d). This scheme provides the following advantages:

- (1) Since  $p$  (table size) is a prime number, the problem of encountering a previously encountered entry in both locating a successor and inserting an item as described earlier does not exist.
- (2) Moving or deleting an item becomes simpler. Figure 3(b) shows the deletion of item  $B$  in Figure 3(a) by replacing the link of  $A$  with the sum of the links of  $A$  and  $B$ .

However, the hybrid of direct chaining and linear probing does not perform as well as the hybrid of direct chaining and linear quotient. Linear probing is less efficient than linear quotient for the problems of primary clustering and secondary clustering (see [9]). These two problems will increase the values of "complete" probe numbers and thus degrade the performance of computed chaining based upon linear prob-

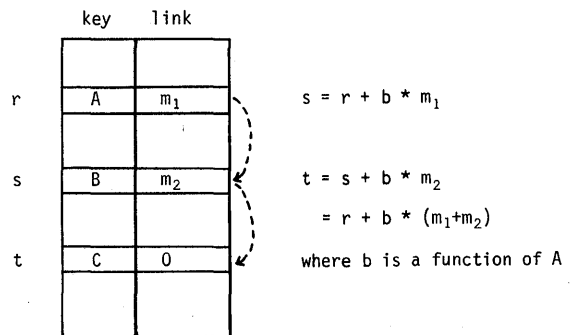


Figure 3(a)—Illustration of computed chaining based upon linear probing.

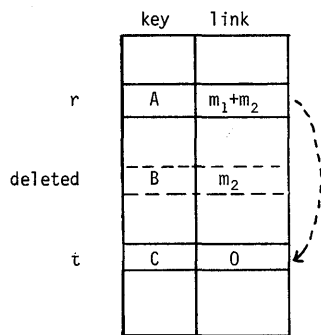


Figure 3(b)—Deletion of item B in Figure 3(a).

ing when the link space is insufficient to store all of the “complete” probe numbers.

PERFORMANCE OF COMPUTED CHAINING

The computed chaining technique described above was tested with a table of size  $n = 997$  using pseudorandom keys. After the table was filled for a specific loading factor ( $\alpha$ ), each item stored in the table was retrieved once and the mean number of probes required to retrieve an item was computed. Table I shows the mean number of probes for retrieval as a function of  $\alpha$  and  $s$  (the number of bits in the link field). The last column of Table I shows  $\text{minis}(\alpha)$ , the minimum value of  $s$  required to store the “complete” probe numbers for a given  $\alpha$ . When  $\alpha$  is smaller so is  $\text{minis}(\alpha)$  because fewer probes are required to find empty cells.

For a given  $\alpha$ , when  $s > \text{minis}(\alpha)$ , the same mean number of probes for retrieval is required as when  $s = \text{minis}(\alpha)$ . In fact, computed chaining needs the same number of retrieval probes as direct chaining when  $s \geq \text{minis}(\alpha)$ . Since  $\text{minis}(\alpha)$  is usually smaller than  $\lceil \log_2 n \rceil$ , where  $n$  is the table size, computed chaining may take less space than direct chaining without loss of efficiency. As shown in Table I, when  $\alpha = 0.99$ , computed chaining requires only eight bits rather than ten bits as in direct chaining.

When  $s < \text{minis}(\alpha)$  for a given  $\alpha$ , more probes are needed to retrieve items because “partial” probe numbers (obtained using the GBD function) are stored in the link fields. Table

I indicates that the performance of computed chaining degrades only slightly as  $s$  becomes smaller. The reason is that most of the “complete” probe numbers are small and thus the number of “partial” probe numbers in the link fields increases slightly as  $s$  becomes smaller. Table II shows the distribution of “complete” probe numbers for  $\alpha = 0.99$ . Sixty-five percent of the “complete” probe numbers are zeros and only five percent require more than four bits.

COMPARISONS WITH OTHER METHODS

As discussed in the previous section, computed chaining, compared with indirect chaining, has the following advantages: (1) it performs as well as direct chaining with less space and (2) it uses even less space by slightly degrading its performance.

Recently several improvements on open addressing which reduce the mean number of probes have been reported. Brent [5] suggested a method of reordering the table when new items are inserted. (A modification of Brent’s method was described in Tharp [6].) Brent’s method requires an average of about 2.49 probes to retrieve an item from a full table ( $\alpha = 1.0$ ). Gonnet and Munro [7] and Mallach [8] presented a better reordering method called “binary tree” hashing which leads to an average of roughly 2.13 probes for a retrieval from a full table. Lyon [9] proposed another reordering method using recursive entry displacements. Gonnet and Munro [7] and Rivest [10] discovered that the problem of reordering the table so as to minimize the average number of probes required for a retrieval is a special case of an assignment problem. Gonnet and Munro [7] reported that the experiment of the optimal reordering scheme produces an average of about 1.83 probes for a retrieval in a full table. Experimental results of several open addressing schemes are given in Table III.

Open addressing, even with optimal reordering, requires more retrieval probes than direct chaining, but does not need extra space as link fields. Computed chaining offers a compromise. It can provide a performance somewhat between direct chaining and open addressing with optimal ordering by using storage between that required by those two methods. As shown in Table III, when  $\alpha = 0.99$  and  $s = 6$ , computed chaining provides better performance than open ad-

TABLE I.—Mean Number of Probes Required by Computed Chaining for Successful Lookup ( $n = 997$ )

$\alpha$	$s = 1$	$s = 2$	$s = 3$	$s = 4$	$s = 5$	$s = 6$	$s = 7$	$s = 8$	$s = 9$	$s = 10$	$\text{minis}(\alpha)$
20	1.090	1.070	1.070	1.070	1.070	1.070	1.070	1.070	1.070	1.070	2
40	1.276	1.214	1.168	1.168	1.168	1.168	1.168	1.168	1.168	1.168	3
60	1.604	1.381	1.271	1.264	1.264	1.264	1.264	1.264	1.264	1.264	4
70	1.881	1.528	1.343	1.323	1.323	1.323	1.323	1.323	1.323	1.323	4
80	2.148	1.715	1.430	1.356	1.356	1.356	1.356	1.356	1.356	1.356	4
90	2.755	2.062	1.666	1.459	1.409	1.408	1.408	1.408	1.408	1.408	6
95	3.294	2.414	1.894	1.600	1.480	1.433	1.433	1.433	1.433	1.433	6
99	4.706	3.330	2.639	2.198	1.829	1.601	1.450	1.449	1.449	1.449	8

TABLE II.—Distribution of Complete Probe Numbers for a 99% Packing Factor in a Table of Size 997

Complete Probe Numbers	Number of Bits to Store Link	Number of Items	Accumulative Percentage
0	0	639	65
1	1	72	72
2 - 3	2	86	81
4 - 7	3	86	89
8 - 15	4	54	95
16 - 31	5	29	98
32 - 63	6	14	99.2
64 - 127	7	6	99.9
156	8	1	100

TABLE III.—Comparison of Mean Number of Probes for Successful Lookup ( $n = 997$ )

$\alpha$	Computed Chaining ( $s=6$ )	Computed* Chaining ( $s=10$ )	Direct Chaining (theoretical)	Pseudochaining ( $s=10$ )	Uniform Probing	Brent's Method	Binary Tree	Lyon**	Optimum
20	1.070	1.070	1.100	1.101	1.115	1.102	1.102		
40	1.168	1.168	1.200	1.211	1.277	1.217	1.217		
60	1.264	1.264	1.300	1.347	1.527	1.367	1.364		
70	1.323	1.323	1.350	1.438	1.720	1.444	-----		
80	1.356	1.356	1.400	1.563	2.011	1.599	1.579	1.49	1.489
90	1.408	1.408	1.450	1.787	2.558	1.802	1.751	1.63	1.610
95	1.433	1.433	1.475	2.023	3.153	1.972	1.880	1.72	1.689
99	1.601	1.449	1.495	2.649	4.651	2.242	2.049		1.785
100			1.500	3.273	6.522	2.494	2.134		1.828

\*These are less than the theoretical expected values  $(1 + \alpha/2)$ . The experiments by Lum, Yuen, and Dodd [11] indicate a similar performance of the linear quotient method.

\*\*For  $n = 4999$ , based on  $I(4)$ ,  $\alpha = .97$  average probes = 1.77  
 $\alpha = .98$  average probes = 1.80

addressing with optimal ordering (1.601 versus 1.785 probes) while using less space than direct chaining (6 bits versus 10 bits per link field). Note that open addressing with optimal ordering requires that the whole table be reconstructed for deletion or insertion of an item and thus is practical only when the table is static. The performance of computed chaining is therefore usually more attractive than open addressing with optimal ordering. Computed chaining is in addition better than Lyon's method of open addressing (1.601 versus 1.80+ probes) which is currently the best method of open addressing not requiring a complete restructuring of the table for an insertion.

The performance of pseudochaining based upon uniform probing [3] and the performance of uniform probing are shown in Table III. The pseudochaining technique can also be applied to improve other open addressing schemes. However, given the same link space, pseudochaining does not perform as well as computed chaining (2.65 versus 1.45 probes for  $\alpha = .99$ ) because the former uses link fields for first overflow items only. As a result, pseudochaining fails to perform the same as direct chaining or computed chaining even if it has sufficient link space to store "complete" probe numbers. Thus pseudochaining is less efficient when compared to computed chaining.

Another advantage of computed chaining, which it inherits from direct chaining, is that items can be deleted immediately without much difficulty. To delete an item from the table, e.g., at location  $r$ , computed chaining will delete the items following in the same chain and reinsert them starting at location  $r$ . In open addressing, however, moving another item in the table to location  $r$  may require reorganization of the table. A common solution in open addressing is to reserve a special key to denote a deleted item so that reorganization of the table can be postponed until a number of deleted items exist. However, the average number of probes required to retrieve non-deleted items will increase if the table contains many deleted items.

## CONCLUSIONS

Computed chaining is a hybrid of several existing methods for collision resolution. It is similar to direct chaining in that

all items with the same home address reside on the same probe chain. Computed chaining resembles open addressing since it uses a function of an item to locate a probe address. And it borrows from pseudochaining the use of a probe number in the link field instead of an actual address.

Practical suggestions on when to use computed chaining conclude the discussion. If sufficient bits in each link field are available to store an actual address, direct chaining remains the preferable method. However, if full-address bits are not available, computed chaining gives results only slightly degraded from those of direct chaining. If only a few bits are available for the link field, computed chaining may still be more efficient than open addressing techniques. From the experimental results ( $\alpha = .99$ ) with computed chaining, sixty-five percent of the items stored in the table required no probe links and ninety-five percent required four or fewer bits to store probe links.

## REFERENCES

1. Knuth, D. E., "The Art of Computer Programming," Vol. III: *Sorting and Searching*, Addison-Wesley, 1973.
2. Maurer, W. D. and Lewis, T. G., "Hash Table Methods," *Computing Surveys* 7, 1 (March 1975), pp. 5-20.
3. Halatsis, C. and Philokyprou, G., "Pseudochaining in Hash Tables," *Comm. ACM* 21, 7 (July 1978), pp. 554-557.
4. Bell, J. R. and Kaman, C. H., "The Linear Quotient Hash Code," *Comm. ACM* 13, 11 (Nov. 1970), pp. 675-677.
5. Brent, R. P., "Reducing the Retrieval Time of Scatter Storage Techniques," *Comm. ACM* 16, 2 (Feb. 1973), pp. 105-109.
6. Tharp, A. L., "Further Refinement of the Linear Quotient Hashing Method," *Information Systems* 4, 1 (1979), pp. 55-56.
7. Gonnet, G. and Munro, I., "Efficient Ordering of Hash Tables," *SIAM J. of Computing*, 8, 3 (August 1979), pp. 463-478.
8. Mallach, E. G., "Scatter Storage Techniques: A Unifying Viewpoint and a Method for Reducing Retrieval Times," *The Computer Journal* 20, 2 (May 1977), pp. 137-140.
9. Lyon, G., "Packed Scatter Tables," *Comm. ACM* 21, 10 (Oct. 1978), pp. 857-865.
10. Rivest, R. I., "Optimal Arrangement of Keys in a Hash Table," *J. ACM* 25, 2 (April 1977), pp. 200-209.
11. Lum, V. Y., Yuen, P. S., and Dodd, M., "Key-to-address Transform Techniques: A Fundamental Performance Study on Large Existing Formatted Files," *Comm. ACM* 14, 4 (April 1971), pp. 228-239.

# A federated architecture for database systems\*

by DENNIS MCLEOD and DENNIS HEIMBIGNER

University of Southern California  
Los Angeles, California

## INTRODUCTION

The contemporary approach to database system architecture requires the complete integration of data into a single, centralized database; while multiple logical databases can be supported by current database management software, techniques for relating these databases are strictly ad hoc. This problem is aggravated by the trend toward networks of small to medium size computer systems, as opposed to large, stand-alone main-frames. Moreover, while current research on *distributed databases*<sup>1,2,3,4,5</sup> aims to provide techniques that support the physical distribution of data items in a computer network environment, current approaches require a distributed database to be logically centralized.

### *Decentralized databases*

A *decentralized database* is a collection of (structured) information, which may be logically distributed, physically distributed, or both. Specifically, it is possible to identify two distinguishable though related aspects of database decentralization:

1. *Logical decentralization* concerns the division of database into components, for purposes of allowing separate control over each component; the control that may be exercised for each component includes specifying the meaning and logical structure of data, describing the accessibility of data items to users, and specifying the form in which users will see the data.
2. *Physical distribution* concerns the allocation of data for storage to the nodes of a network or other assembly of interconnected computer system components.

A comprehensive approach to decentralized database management must address both the issue of logical decentralization as well as the issue of physical distribution.

### *The need for logical decentralization*

Traditionally, a database is viewed as a complete and total integration of the data associated with a family of related, though distinct, applications. A database has associated with it a single structural specification: its conceptual/logical *schema*. Users and application programs manipulate the data by performing operations phrased in terms of the schema. The database is then physically realized by a particular *physical design*, which is a collection of storage structures and access methods that actually implement the schema.

In contrast to the approach in which data files are closely associated with application systems and isolated from one another, the "integrated database" approach is founded on the principle of logical centralization. The complete centralization of data at the logical level has many benefits associated with it; in fact, these benefits are largely responsible for the great success of the "database approach" during the past decade:

- Complete integration provides a global view of the data resources of an organization, and provides a basis for the resolution of conflicts; the importance of the database as an organizational resource is recognized.
- By constructing a single integrated database, the amount of data redundancy in the overall information system is significantly reduced; this reduction in redundancy diminishes the opportunities for data inconsistencies and related problems.
- Centralization enables the more ready implementation of database applications that require data from several sources.
- Logical centralization of a database allows uniform modes of access and usage to be established for all the data.

While logical database centralization has important associated benefits, it does impose certain limitations on database-intensive information systems. Specifically, it is often extremely difficult to completely integrate applications that are related, yet separate; integration may go too far in tightly coupling together aggregates of data that ought to retain some individual autonomy.

\* This research was supported, in part, by the Joint Services Electronics Program through the Air Force Office of Scientific Research under contract F44620-76-C-0061.



In the conventional view of database design, based on the concept of complete logical centralization/integration, a central authority is responsible for designing and maintaining "the" database; this authority, be it a single person or a group of individuals acting together, is usually called the *database administrator* (DBA). The DBA maintains control over all the data, and is responsible for determining and adjudicating the disparate needs of the various database applications and users. In such an approach, the ultimate providers and users of the data must relinquish their authority over it to the DBA; this raises a number of concerns:

- Users are often hesitant to entrust their data to an external authority, despite any assurances they receive; experience has shown that these reservations may indeed be valid.
- The DBA is charged with developing a unified specification of the content and meaning of a database. In practice, this is a difficult task, since various database users will have different perceptions of the data.
- In the process of selecting a physical design for a database, the DBA must ascertain the global usage patterns and response requirements, and then select the alternative physical design that provides the best overall performance. While this approach may indeed best serve the overall organization, it may not be well suited to the needs of the principal users of a given portion of the database. A compromise physical design that attempts to satisfy all database users may in fact satisfy none of them.
- Charged with the problem of serving as a liaison between the database and all of its users, the DBA can easily become a bottleneck. All requests for database extensions and revisions are funnelled through the DBA; this indirection can and often does introduce serious delays and inconsistencies, particularly as the complexity of a database grows.
- New databases are often created as combinations of old databases; that is, it is not always true that a database is designed in strictly top-down fashion. In consequence, it is often very difficult to try to totally integrate two related, but separate, databases into a unified whole.

### *Distributed databases*

One particular approach to database decentralization is commonly called *distributed database systems*. In this approach, a single logical database schema is defined, which describes all the data in the database system; the physical realization of the database is then distributed among the computers of a network. The physical data of a distributed database can be divided in three ways:

1. *partitioned*, where no data is duplicated,
2. *fully duplicated*, where all data is duplicated in every computer,
3. *partially duplicated*, where some data is duplicated at some computers.

There are two main advantages to distributed databases:

1. A distributed database system is potentially more efficient than a physically centralized database system, because the data can be placed close to where it is needed. If it is needed at two or more places, then it can be duplicated.
2. If data is duplicated, then a distributed database is potentially more reliable than a physically centralized database, because even if one computer fails other computers in the network may continue to operate.

Although there are advantages to distributed database systems, there are also a number of difficult design issues associated with them:

- How can the data be optimally allocated to the computers to minimize some cost, such as access time or physical storage space?
- How can a distributed database continue to operate after the failure of a computer?
- How can duplicated data be kept consistent?

In response to the observation that decentralized computing systems are of increasing general importance, and the realization that logical centralization of a database (with or without physical decentralization) has many problems in practice, it is clear that a fresh approach is required. The goal of this new approach must be to serve as a compromise between total integration/centralization and the disorganization of completely diffused and decentralized databases. The key to successfully realizing this goal is to balance the need for decentralization and the largely conflicting need for effective sharing of information.

### A FEDERATED APPROACH TO DATABASES

The approach to database decentralization advocated here is termed *federated databases*; the basic idea of federation was introduced by Hammer and McLeod.<sup>6</sup> A federated database consists of a number of logical *components*, each having its own logical/conceptual schema (*component schema*). These components are related, but independent, and they may or may not be disjoint. Typically, a component of a federation corresponds to a collection of information needed by a particular application or a set of closely related applications.

All of the components in a federation are tied together by one or more *federal schemas* that express the commonality of data throughout the federation; these federal schemas are used to specify the information that can be shared by the federation components, and to provide a common basis for communication among them.

Database system users and application programs manipulate a database by issuing *transactions*, viz., operations that retrieve information from or modify information in a database. As a database user or application program is most commonly affiliated with a single component of a federation, that

user (or application program) normally issues transactions that can be performed within the local component. This property may be termed locality of reference and is fundamental to federated database systems.

On occasion, a user of component C1 may need to issue a transaction that involves data that belongs to another component, C2 (or several other components). In this case, the user consults a federal schema to find the necessary data; this reference can be explicit or implicit (i.e., the user may either refer to the data in the context of the federal schema, or may refer to it as local derived data (in which case the derivation specification must have already been provided)<sup>7</sup>. A transaction involving nonlocal data is processed by issuing a request to the *federal controller*, which issues the necessary instruction to C2 to actually provide the necessary data. While transactions involving local data execute with all possible speed, transactions that require non-local data are in general substantially less efficient, because the federal controller must intervene to perform data movement and translation. The federal controller is thus an important part of a federated database system, playing the role of coordinator and translator.

In the federated approach, the (conceptual/logical) schema of each component is defined by a *component DBA*. A component schema is designed to suit the users and applications of the component; and, the physical design used to implement a component schema is developed (and will evolve) so as to best satisfy the performance requirements of these local users. In this way, the principal goal of each component is to satisfy its most frequent and important users (viz., the local ones).

All federal schemas are defined and controlled by the *federal DBA*. Each federal schema is a virtual one, in the sense that there does not exist a physical database that corresponds to it; rather, a specification is provided that describes how the federal schema constructs are materialized from data maintained by the individual components. In particular, each component defines a subset of its component schema as available to the federal schema(s).

The duties of the federal DBA supplement, rather than conflict with, the activities of the component DBAs. The principal responsibility of the federal DBA is to define the federal schema(s), relate them to the component schemas, and define the interface that each component must provide. The federal DBA is also responsible for determining how logical redundancy in the federation ought to be handled: in some cases, it is appropriate for a single component to take responsibility for it; in other cases, it is better for each component to maintain its own version (with a variety of possible consistency restrictions established to ensure that the various versions remain appropriately related, e.g., the same). The choice may be determined for reasons of efficiency, reliability, or requirements of components that need to access the data.

In addition to directly accommodating logical database decentralization, the federated architecture also enhances the evolvability of a database. A federation evolves either by changes to components or changes to a federal schema. As long as a component continues to support its interface

to the federation, it is free to change either its physical structure or its logical structure without affecting other components (except possibly with regard to performance). The federal schema can change for one of four reasons:

1. a deliberate policy decision to change the federal schema,
2. a radical change in a component that requires a change in its interface to the federation,
3. adding a new component to the federation,
4. deleting a component from the federation.

Changes that enlarge or restructure the federal schema, such as by the addition of components, will impact components to the extent that they must accommodate the new information in the federal schema. Changes that actually remove information, such as the deletion of a component, in general require other components both to accept an altered federal schema and to redesign transactions that access the deleted information.

In sum, in the federated approach, primary control over a database component resides with its principal maintainers and users, but adequate centralized authority is exercised in order to ensure appropriate levels of sharing, data compatibility and data consistency. Each federation component can determine how to optimize its part of the database according to its own needs, and can decide what information should be made available to other components. Sharing of information is accommodated by the federal schema, and conflicts are resolved by the federal DBA. Finally, the federated database architecture is based on the observation that many contemporary integrated databases are actually better suited to partial decentralization than complete centralization; for example, despite the availability of an integrated database, it is often the case in practice that the functional units of an organization make use of only a subset of the total schema and a limited portion of the data; in such cases, the remainder of the database can actually be a burden to a user.

## DESIGN ALTERNATIVES FOR FEDERATED DATABASE SYSTEMS

Any design for a federated database system must deal specifically with the following issues:

- the precise structure of the federation (viz., the number and organization of the federal schemas, and their relationship with the component schemas),
- the handling of physical data storage and access in the federation,
- the specific approach to the operation of the federal controller,
- the component facilities to support interaction with the federation.

These four important design issues are specifically examined immediately below.

### *Logical distribution*

The logical distribution of a federation determines the ease with which changes to the schemas can be made and ease of maintaining the federal schemas. There are four principal logical distribution alternatives, with differing ability to handle change and maintenance:

1. The first logical distribution strategy involves a single, global, federal schema derived from all the components. This structure is simple for the federal DBA to maintain, because there is only one federal schema. But such a comprehensive federal schema is difficult for the DBA to design, because it must reflect all the desired interactions between components. In addition, components are restricted from making radical changes in their component schemas because it may require changes to the federal schema. Components may be prohibited from seceding from the federation, because that may also require changes to the federal schema.
2. An alternative distribution uses a separate federal schema for each pair of components. In the worst case of  $n$  components totally interconnected, there will be  $n(n-1)/2$  federal schemas. Defining and maintaining this number of federal schemas may well place an intolerable burden upon the federal DBA, particularly for a large  $n$ . However, it may be that for a given federation only some small portion of the possible interconnections is needed. Each pairwise federal schema is simpler than a global schema, since fewer components are interacting. In this pairwise federal schema approach, adding or removing components is simple: the component and its federal schemas are removed.
3. A third logical distribution alternative is to associate a federal schema with each component, for use by all other components. Each component maintains two interfaces, a local one for its users and another one for use by all other components. In this approach, it is easy to add or remove components, and the number of federal schemas is equal to the number of components.
4. A final logical distribution strategy is a variation of the global distribution strategy: instead of a single global federal schema, there are several federal schemas arranged in a hierarchy. In this organization, the components are separated into disjoint sets with a federal schema for each such set. The federal schemas at the first level are partitioned into groups, and a second level set of federal schemas is defined upon the sets of first level federal schemas. This continues until a single federal schema (designated the root) is constructed. The result is a tree of schemas; the leaves are the component schemas and all interior nodes are federal schemas. In this approach the effects of adding or removing a component may be limited to some subtree of the hierarchy. Clearly, in this strategy, the simplicity of the federal schema hierarchy is determined by the criteria used to structure the tree.

Also at issue in logical distribution is the nature of the view seen by a user associated with a given federation component. A user associated with a given component must be able to access both local data, through the local schema, and non-local data through a federal schema. The local data is accessed by the normal mechanisms of the system (i.e., a data manipulation facility/language or programming language interface). Access to non-local data depends upon the user's view of the federation. At one extreme, the federal schema is integrated with and extends the local schema in such a way that the user cannot tell if he is accessing local data or non-local data. At the other extreme, the federal schema is separate from the local schema, although not necessarily disjoint; in this situation, the user must specifically address his request to the local schema or the federal schema.

Complete integration makes it simple for a user to express a database transaction, since both local and non-local data look the same; the principal problem with this approach is that the user cannot directly observe that a potentially expensive non-local reference may be required to process the transaction. When there is no integration, the user must perform extra steps to retrieve non-local data, which may then be combined with a manipulation of local data. In this case, the user knows that a potentially expensive non-local reference is needed. There is, of course, a viable middle ground between these extremes, in which the user sees two separate schemas, (component and federal) and the database transaction processor accepts combined references to both local and non-local data. In this way, the user knows a costly non-local reference is being made, but the details of accessing are delegated to the database system.

### *Physical distribution*

The federated database architecture does not assume that a database will actually be supported in a distributed environment; that is, it is not assumed that the database is to span a number of nodes in a computer network. A federated database could well be implemented on a single computer. However, there are advantages to physically distributing data:

1. to achieve better performance and allow higher degrees of concurrency by placing data close to its principal sources and users,
2. to provide a higher degree of reliability and survivability by redundantly storing data items.

The federated database architecture directly addresses the first of these two main goals; the concept of locality of reference is key in the federated architecture. Moreover, the federated architecture provides a basis, through the federal schema and federal DBA, for establishing a policy for redundant data storage.

As noted above, one of the main principles of the federated database architecture is that the responsibility for storing and supporting physical access to the data in each compo-

ment of a federation is the responsibility of that component. Thus, the most general approach might be to allow each component to choose its own method for storing data; if a computer network is being used to implement a federated database, each component may distribute its own data throughout the network.

However, intolerable complexity may result from complete flexibility for physical distribution along with complete flexibility for logical decentralization. Moreover, logical decentralization and physical distribution are not orthogonal issues. In consequence, it is appropriate in many cases to directly combine logical decentralization with physical distribution. In this approach, if a computer network is available for database implementation, then each federation component is allocated to a node in the network. The matching of a federation component to a node in a computer network provides a direct and natural way to implement a database that can be both logically decentralized and physically distributed.

Another aspect of physical distribution is the control of duplicate data. When two components contain duplicate information in their schemas, the federal DBA must decide how that data is to be handled in the federation. Duplicate data can be eliminated from the physical level of the federation by selecting one copy as the official copy; all references to a specific data item then refer to the official copy.

If duplicate data is retained, and it is desired that it be kept consistent (i.e., that all copies ultimately reach the same value after database modifications cease), then it is possible to apply the techniques developed for controlling duplicate data in distributed databases.

A number of control algorithms for maintaining consistency in distributed databases have been developed;<sup>5</sup> the algorithms that support partially duplicated data are directly relevant to federated databases. The proposed algorithms for maintaining the consistency of partially duplicated data are complex, since they attempt to keep all duplicate data as current as possible. This is important in a distributed database system so that the users continue to see a logically centralized database. However, complete consistency is not necessarily important for federated databases, because they are logically decentralized. In consequence, it may be possible to apply looser and simpler algorithms for controlling duplicate data in the federated environment.

#### *Federal controller operation*

A federated database requires a control component not present in conventional (centralized) database system: the federal controller. As described above, the federal controller performs the bulk of the transformations necessary to satisfy a request from a component for information described in a federal schema (and that is contained in another component); the request takes the form of a specified transaction. The federal controller must perform a sequence of seven steps for each such request/transaction:

1. The transaction is checked for legality against the federal schema. The access rights of the requester are also verified at this time.
2. The transaction is decomposed into a collection of simpler *target transactions*, each of which can be ultimately satisfied by a single *target component*. The target component is the component that supports that part of the federal schema referenced by the target transaction.
3. Each target transaction is translated from a reference to the federal schema to a reference to the target component schema.
4. The target transactions are sent to the corresponding target components for processing.
5. The federal controller waits for all the target transactions to be processed, and then the controller collects the results.
6. The results are translated from target schema form back to federal schema form.
7. The translated results are combined and returned to the requester.

Steps five through seven can be performed in either *set-at-a-time* or *element-at-a-time* fashion. In set-at-a-time processing, the federal controller collects the results from all of the target components into a single result set, which is then returned to the requester. In element-at-a-time processing the federal controller translates and returns to the requester each element of the result as it is made available by a target component. The choice between set-at-a-time and element-at-a-time processing should be made based on storage cost and communication cost information.

The federal controller can itself be either centralized or distributed. If a computer network is used for implementing a federated database system, then there are three main approaches to federal controller placement:

- The federal controller resides on a special node of the network, i.e., one which does not also contain a component. This approach has the advantage of isolating the federal controller, and the controller node need possess only the computational power necessary to perform the controller's functions. In this approach, it is also possible to easily replace the controller, should it fail. The disadvantages of a special controller node include the need for additional hardware, and the potential problem of a system performance and reliability bottleneck.
- The federal controller can be co-located on a node with one of the components of the federation. This saves the cost of extra hardware, at the cost of possible competition for node resources with the component controller. The controller can also be made to migrate from one component node to another, should a node fail or a performance improvement be possible by shifting control.
- The federal controller can be distributed, in which case a part of the controller is located at every node (or some subset of the nodes). This has advantages for

reliable operation, but the coordination of all the controllers is a difficult problem.

The choice between a centralized or a distributed federal controller must be made in the context of the relative complexity of the algorithms for supporting coordination (analogous to work on distributed database control algorithms<sup>5,8,9,10,11,12</sup>), and the relative storage and communication costs involved.

### Component control

In most respects, the control aspects of a component of a federation are the same as those for a centralized database system; but since a component is part of a federation, it must support an appropriate interface to the federation. In particular, there are several important issues that a component must address, vis-a-vis its interaction with the federation:

- The component must allow for concurrent access to its data, because while a local user is accessing some part of the component data, some other component may be attempting to simultaneously access the same data (through the federal controller). If the component already has the capability for concurrency control for local users, then requests by the federal controller present no difficulty. Otherwise, the component software must be augmented by software to control the simultaneous access attempts.
- The component software must provide for communicating results back to the federal controller, on either a set-at-a-time or an element-at-a-time bases. Set-at-a-time processing requires bulk transfer of information to the federal controller, and element-at-a-time processing requires the buffering of the results at the component followed by single element transfers to the federal controller.
- The component must recognize locally-issued transactions that require accessing the federal schema, and forward an appropriate request for processing to the federal controller. When the federal controller returns the result of a transaction, the component combines the results from the federal controller with the results of any portion of the transaction that referenced data local to the component.

In sum, it is the combined functioning of the components and the federal controller that allows a federated database system to effectively support information sharing and the decentralization of data.

### SUMMARY

A federated architecture for database systems has been presented, which supports the logical decentralization of databases, and provides a basis for database physical distribution (in a network of computer systems). The federated

architecture responds to a number of problems associated with the complete centralization and integration of database systems (as detailed above).

A federated database consists of a number of logical components, each having its own user-level structural specification (component schema). The components of a federation are related, but independent, and they may or may not be disjoint. Typically, a component corresponds to a collection of information needed by a particular user or application. The components in a federation are tied together by one or more federal schemas that describe the data that is to be shared by the various federation components, and provide a common basis for communication among them. A federal controller, which is an essential constituent of a federated database system, supports communication and translation of data among federation components, based on the federal schema(s).

In this paper, a number of design issues and alternatives for federated database systems have been reviewed. Alternative logical distributions and physical distributions were described, and the issues relevant to the operation of the federal controller and federation components discussed. We are presently developing a specific design approach based on the principles described in this paper.<sup>7</sup> A critical aspect of our present approach is the use of a meaning-based (semantic) database description and structuring formalism (database model) (such as those described in<sup>13,14,15,16,17,18,19,20,21</sup>) to specify the component schema interface with the federal schema(s).

### ACKNOWLEDGMENTS

The authors are grateful for the very helpful efforts of Michael Hammer of MIT, a co-developer of the federated database concept. Gerald Short of TRW and the CADAM group of the Lockheed California Company (under Don Kawamoto) have provided additional motivation for the federated architecture.

### REFERENCES

1. Champine, G., "Six Approaches to Distributed Databases," *Datamation*, Pages 45-48, May 1977.
2. Lien, Y. E. and Ying, J. H., "Design of a Distributed Entity-Relationship Database System," *Proceeding of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1978.
3. Miller, M., "A Survey of Distributed Data Base Management," *Information and Management*, Pages 243-264, 1978.
4. Ramamoorthy, C. V. and Wah, B. W., "Data Management in Distributed Data Bases," *Proceedings of National Computer Conference*, New York, NY, 4-7 June 1979.
5. Rothnie, J. B. and Goodman, N., "A Survey of Research and Development in Distributed Database Management," *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
6. Hammer, M. and McLeod, D., *On the Architecture of Database Management Systems*, Technical Report 79-4, Computer Science Department, University of Southern California, Los Angeles CA, April 1979.
7. McLeod, D., *An Approach to Database Decentralization*, Technical Report, Computer Science Department, University of Southern California, Los Angeles CA, 1980 (to appear).

8. Bernstein, P. A., Rothnie, J. B., Goodman, N., and Papadimitriou, C. D., "The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (the Fully Redundant Case)," *IEEE Transactions on Software Engineering*, Volume SE-4, Number 3, May 1978.
9. Bernstein, P. A. and Shipman, D., "A Formal Model of Concurrency Control Mechanisms for Distributed Database Systems," *Proceedings of Third Berkeley Conference on Distributed Data Management and Computer Networks*, Berkeley CA, 29-31 August 1978.
10. Garcia-Molina, H., "Performance Comparison of Two Update Algorithms for Distributed Databases," *Proceedings of Third Berkeley Conference on Distributed Data Management and Computer Networks*, Berkeley CA, 29-31 August 1978.
11. Stonebraker, M. and Neuhold, E., "A Distributed Database Version of INGRES," *Proceedings of Second Berkeley Conference on Distributed Data Management and Computer Networks*, Berkeley CA, May 1977.
12. Thomas, R. H., "A Majority Consensus Approach to Concurrency Control," *ACM Transactions on Database Systems*, Volume 4, Number 2, June 1979.
13. Buneman, P. and Frankel, R. E., "FQL—A Functional Query Language," *Proceedings of ACM-SIGMOD International Conference on the Management of Data*, Boston MA, 30 May-1 June 1979.
14. Chen, P. P. S., "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Transactions on Database Systems*, Volume 1, Number 1, Pages 9-36, March 1976.
15. Codd, E. F., "Extending the Database Relational Model," *ACM Transactions on Database Systems*, vol. 4, no. 4, December 1979.
16. Hammer, M. and McLeod, D., "The Semantic Data Model: A Modelling Mechanism for Database Applications," *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Austin TX, 31 May-2 June 1978.
17. Hammer, M. and McLeod, D., *SDM: A Semantic Database Model*, Technical Report, Computer Science Department, University of Southern California, Los Angeles CA, 1980 (to appear).
18. McLeod, D. and King, R., "Applying a Semantic Database Model," *Proceedings of International Conference on the Entity-Relationship Approach to Systems Analysis and Design*, Los Angeles CA, 10-12 December 1979.
19. Shipman, D., "The Functional Data Model and the Data Language DAPLEX," *ACM Transactions on Database Systems*, 1980 (to appear).
20. Smith, J. M. and Smith, D. C. P., "Database Abstractions: Aggregation and Generalization," *ACM Transactions on Database Systems*, Volume 2, Number 2, Pages 105-133, June 1977.
21. Su, S. and Lo, D., "A Semantic Association Model for Conceptual Database Design," *Proceedings of International Conference on the Entity-Relationship Approach to Systems Analysis and Design*, Los Angeles CA, 10-12 December 1979.



## Data Base Architecture and Management

The development of integrated data bases and Data Base Management Systems (DBMS) has resulted in recognition that these alone do not insure information system effectiveness or user satisfaction, nor do they control the cost of handling data. Meeting these goals ultimately rests with improving the organizational management/administration and understanding of "data" and "information" and their relationship.

We must realize that DBMS is only one component of Data Management Systems (DMS) consisting of data resource directory, data dictionary, query language, report generator, and usually a teleprocessing monitor. Data base technology has

focused, for the most part, on the constructs of data and their manipulation.

The sessions in this technical area discuss current trends and advanced approaches to data base architecture, data base management systems, "natural language"-based query languages, and implementing a data management plan.

If we recognize that data and information are related but not exactly the same, then we face the problem of redesigning our "data systems" to provide "information."

The four sessions in this technical area explore a broad spectrum of new avenues for making data base systems more useful in providing "information" to users through new architecture and query approaches.

A data management implementation plan that emphasizes the pitfalls and how to resolve them is presented in the session chaired by Linda Taylor. This part of the session is followed by a description of two new user-oriented approaches to accessing data, assuming you have now implemented a sound data management plan.

Dennis McLeod chairs a session discussing recent research at UCLA, IBM, US-Berkley, and Computer Corporation of America in applied data base technology. The results of this work are expected to affect the design and architecture of future data base management systems.

E. L. (Ted) Glaser's session includes a panel of eminent authorities in the area of DBMS and data base architecture design, describing and then "debating" the advantages and disadvantages of various DBMS and data base architecture.

The session chaired by Norm Sondheimer features an illustrious group of panelists who will examine current and predict future successful and problematical approaches to natural language access to a data base system.



Linda Taylor  
*Area Director*





# Definition of database transactions by the casual user

by FRED J. MARYANSKI\* and C. STEVEN ROUSH\*\*

*Computer Science Department  
Kansas State University  
Manhattan, Kansas*

## INTRODUCTION

Database management systems have been in general use for more than a decade. However, only recently have advances in technology and reduction in cost made such systems feasible for the small enterprise. Experience with database systems over the years indicates that the definition of a schema, design and implementation of a set of application programs, and maintenance of the system are far from trivial tasks. Consequently, the ability to perform effectively any of the above three tasks has become a highly marketable skill.

The need to make database systems accessible to the casual, or nonprogramming, user was first addressed by Codd [9] and has been the subject of considerable recent work [6, 7, 10-13, 15-17, 19-21, 23-26]. Much of the effort in this area has centered upon the development of interactive query languages for the specification of ad hoc database transactions. In this document, we report the results of an effort to permit the casual user to specify a complete set of transactions for an enterprise. This transaction definition subsystem produces a set of "canned" transactions to which the user supplies any runtime parameters.

The transaction definition subsystem is one unit in an application development system designed to facilitate database utilization by the nonprogramming users. The goal of the project is to create a database system that can be comprehended and effectively utilized by a user whose enterprise cannot support a data processing professional. It is assumed that the user of the application development system is unfamiliar with the concepts of database management but highly knowledgeable concerning the data of the enterprise. The system is highly interactive and relies upon the user to supply all information on the data items, relationships among data items, and operations on the database.

In a prior effort a subsystem that permits a casual user to interactively create a third normal form schema has been developed [3, 14, 22]. The outputs of this subsystem are used in the transaction definition process. Figure 1 depicts the present status of the application development system.

The remainder of this paper concentrates upon the trans-

overview of the methodology employed to define the transaction definition subsystem. The next section contains an actions. The structure of the subsystem is elaborated upon in the third section by describing the processing and interaction that take place. The concluding section describes possible future efforts in this area. A sample transaction definition session is included in the appendix.

## OVERVIEW

The function of the transaction definition subsystem is to receive a description of a document and through interaction with the user, create a transaction which generates the document. The definition of a transaction is dependent upon the presence of a hierarchical description of the document that the transaction is to produce and a third normal formal schema which embodies all noncomputed data items on the document. The hierarchical description and schema are outputs of the data definition subsystem which must be completed prior to the execution of the transaction definition subsystem.

The document descriptor indicates all data items on the document and the hierarchical structure of the document. The hierarchical structure is a tree representing the logical organization of the document. The nodes of the tree are the unique lines of the document. Figure 2 illustrates the structure of the document descriptor. Detailed information on the construction of the document descriptor can be found in Reference [14].

The feature that distinguishes this approach to transaction definition from other methods is that the user need not be aware of the structure of the database when the transactions are defined. However, we must emphasize that the user is required to be thoroughly familiar with the meaning and organization of the data on the documents of the enterprise.

The transaction definition subsystem is highly interactive. The user is questioned in order to obtain information on the semantic nature of the transactions. Figure 3 depicts the structure of the transaction definition subsystem. The function of each of the modules is explained in the succeeding section.

The output of the transaction definition subsystem is a modified form of SEQUEL2 [5] code intended for execution by a relational database management system.

\* Address: Digital Equipment Corp., Maynard, MA 01754

\*\* NCR Corp., Wichita, Kansas 67226

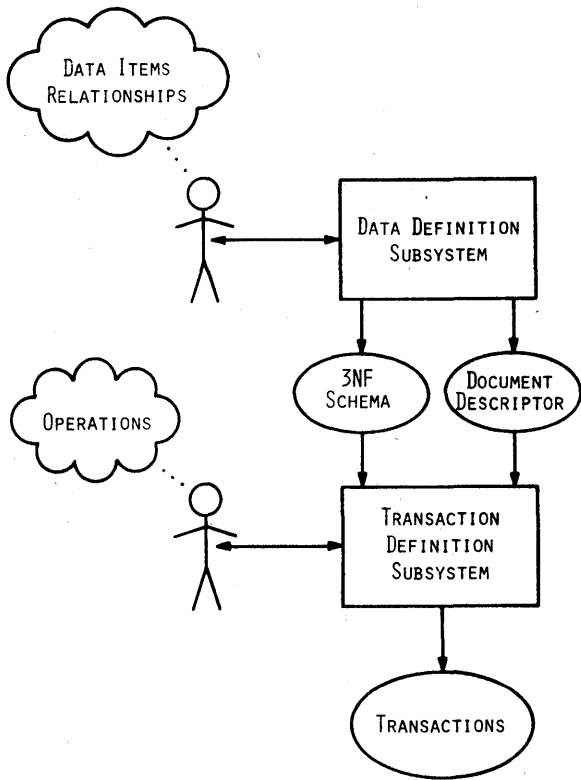


Figure 1—Application development system

STRUCTURE

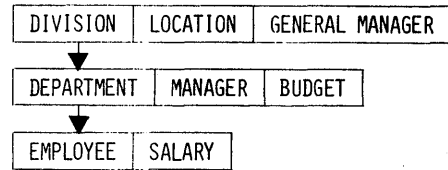
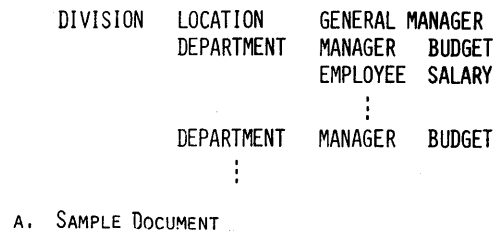
In this section we explain the workings of the transaction definition subsystem by detailing the functions of the modules pictured in Figure 3. For purposes of the presentation, the operation of the subsystem is explained initially for a retrieval transaction. The last portion of the section indicates the differences involved in the definition of an update transaction. A complete description of the transaction definition subsystem is available in Reference [18].

Initialization

The schema and document descriptor are read as input, and various internal tables are constructed using this information. The HANDLE-TRANSACTIONS module calls the PROCESS-LINE module in an iterative manner in order to generate the code for each document in the user's application system.

Identification of primary relation

A key concept of the transaction definition subsystem is the identification of the relation closest in terms of domains to the content of each line of the document to be produced. This relation is termed the *primary relation* for the line. The primary relation is used as the starting point for the navi-



B. HIERARCHICAL STRUCTURE

NAME	LEVEL	GROUP	FLAG
DIVISION	1	110	1 (KEY)
LOCATION	1	110	0 (NON-KEY)
GENERAL MANAGER	1	110	0
DEPARTMENT	2	210	1
MANAGER	2	210	0
BUDGET	2	210	0
EMPLOYEE	3	310	1
SALARY	3	310	0

C. DOCUMENT DESCRIPTOR

Figure 2—Document descriptor example

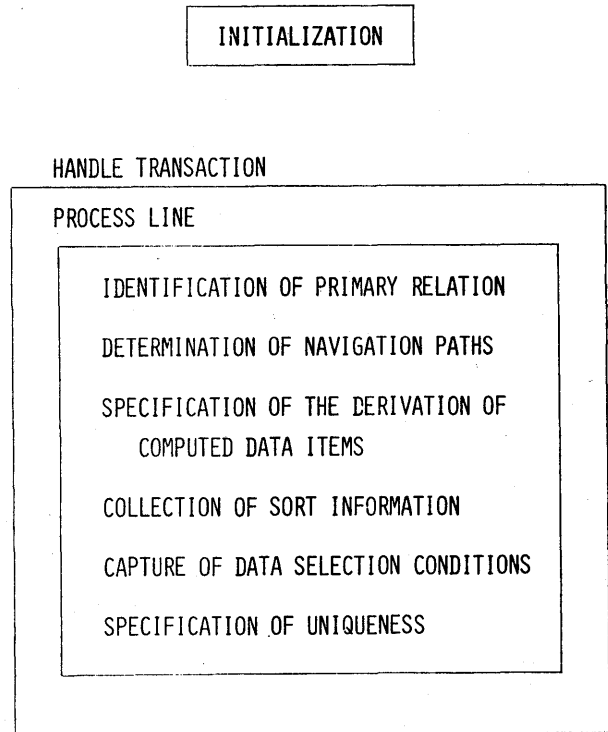


Figure 3—Structure of transaction definition facility

gation paths that link all data on the line. The determination of the primary relation is made without interaction using information in the schema and document descriptor. Figure 4 is a high level description of the primary relation identification algorithm. The primary relation is the starting point for all further processing by the transaction definition subsystem.

As indicated in the high level description, the algorithm compares the contents of the lines and the relations. It is possible that two or more relations may be equally close, according to the metric of the algorithm, to a given line. In this situation the program maintains multiple primary relations and begins navigation from all primary relations in order to cover all elements in a line of a document.

#### *Determination of navigation paths*

In order to produce a transaction without requiring that the user have a knowledge of the schema, the system must be able to determine navigation paths from the primary relation to relations containing all data items in the line being processed. The algorithm for the determination of a navigation path for retrieval operations is portrayed in Figure 5. The algorithm involves searching through relations begin-

#### DETERMINE A REASONABLE SET OF CANDIDATE DETERMINANTS OF THE LINE

1. ELIMINATE ELEMENTS NOT MARKED AS POSSIBLE DETERMINANTS BY THE USER
2. ELIMINATE ELEMENTS NOT KEY TO ANY RELATION
3. IDENTIFY ELEMENTS WHICH ONLY APPEAR IN RELATIONS AS KEYS
4. ELIMINATE ELEMENTS WHICH NEVER APPEAR IN A CONCATENATED KEY WITH THE ABOVE ELEMENTS

#### DETERMINE A SET OF CANDIDATE RELATIONS

1. START WITH EVERY RELATION WITH ANY CANDIDATE DETERMINANT IN ITS KEY
2. ELIMINATE THOSE RELATIONS WITHOUT FULLY COVERED KEYS
3. ELIMINATE ANY CANDIDATE RELATION WHICH CAN BE COVERED BY ANOTHER

REMAINING RELATION IS THE PRIMARY RELATION

Figure 4—Primary relation algorithm

#### NAVIGATION

```

FOR EACH ELEMENT INVOLVED
  IF FOUND IN PRIMARY RELATION
    STOP--THAT'S IT
  ELSE
    IF IT APPEARS IN ONLY ONE RELATION AS NONKEY
      STOP--THAT'S IT
    ELSE
      FOR EACH RELATION IN WHICH IT APPEARS AS NON-KEY
        IF ALL THE KEYS OF THAT RELATION ARE NOT
          IN THE LINE
            ELIMINATE THAT RELATION FROM CANDIDACY
      ENDIF
    ENDLIST
  IF ONLY ONE CANDIDATE LEFT
    STOP--USE IT
  ELSE
    CAN'T BE DETERMINED
  
```

Figure 5—Algorithm for navigation paths

ning at the primary relations until all data items for the line have been reached. Although the implementation is different, the algorithm for the determination of the navigation paths is conceptually similar to Bernstein's membership algorithm [1, 2].

#### *Specification of the derivation of computed data items*

During the execution of the data definition subsystem, the user has identified all data items which are computed and consequently not included in the schema. This information is preserved in the document descriptor which is an input to the transaction definition subsystem. The user is requested to indicate the data items used in the derivation of the value of each computed data item. In the current version of the prototype only the operands, but not the operators, of the expression for the computed data item's value need be specified. Alternatives for the capture of the complete expression for a computed data item are being investigated.

#### *Collection of sort information*

The user is asked to supply any sort keys in major to minor order and the sorting sequence for each key.

#### *Capture of data selection conditions*

The user is requested to indicate if he wishes all instances of the data items on the line or if he wishes to apply selection criteria to the data. Again, a reasonably straightforward interaction, as illustrated in Figure 6, is utilized to capture the data selection conditions.

Another limitation of the prototype affects the operands

DO YOU WANT THIS TRANSLATION TO INVOLVE EVERY  
OCCURRENCE OF THE DATA FOR THIS LINE? (Y/N)  
( 'N' IF THERE IS ADDITIONAL SELECTION CRITERIA)

(ACCEPT ANSWER)

OF THESE ELEMENTS IN THE LINE

(DISPLAY LINE)

WHICH IS INVOLVED IN THIS CONDITION?

(ACCEPT ITEM USED IN SELECTION)

PLEASE ENTER CONDITIONAL OPERATOR (=, <, >, ETC.)

(ACCEPT OPERATOR)

FOR NOW, THE 'RIGHT-HAND SIDE' WILL COME FROM THE  
CRT

Figure 6—Capture of data selection conditions

of the selection expressions. In the case of selection criteria, one operand must be a data item on the line being processed. In the prototype, the other operand is restricted to being a value accepted from the keyboard. Many possibilities exist for the source and format of the second operand. The most difficult situation arises when the second operand must be retrieved from the database. The main difficulty here is the specification of the source of the operand by the user. This problem is currently under study.

#### *Specification of uniqueness*

Depending upon the selection criteria, an operation may retrieve tuples containing duplicate values. In some situations, the user may desire to observe only unique tuples satisfying the selection criteria. This option is provided interactively to the user. Based upon the response to the uniqueness question, a command that will retrieve either unique or duplicate tuples is generated.

#### *Update transactions*

As shown in Figure 7, the overall structure of the algorithms for the production of transactions that write to the database is similar to that of the algorithm for read-only transactions. In a retrieval transaction, it is necessary to establish the existence of navigation paths that reach at least one occurrence of each data item to be retrieved. When a data item is to be written (stored, updated, or deleted), all occurrences of that data item must be located.

In a third normal form database, the problem of multiple occurrences exists only for keyed data items. The existence

of a third normal form schema implies that for a given set of keys, the tuple containing that data item must be uniquely determined [4, 8]. However, keyed items may exist in multiple relations. Therefore, the navigation path routines include the ability to locate all instances of keyed data items. Stored and deletion transactions will always operate upon keyed data items. At the present time, the ability to update a keyed data item is not provided.

#### *Example*

The appendix contains a comprehensive example of the input, user interaction, and output of the transaction definition subsystem. The information in the appendix should be self-explanatory. The only non-SEQUEL2 statements produced as output are the SELECT(NEXT) and SELECT(ALL) which are iterative retrieval statements. A SELECT(NEXT) statement in effect defines a loop in which one tuple is retrieved and then, in this example, the following ACCEPT and SELECT statements are executed. A SELECT(ALL) statement causes all tuples that satisfy the selection criteria to be retrieved. In this example, all grades will be listed on a student-by-student basis.

## CONCLUSION

#### *Summary*

The application development system described here is oriented toward the casual user or the small businessman with the need for a computer but not a programming staff. The system captures the data from the user and then interactively synthesizes a third normal form schema and the transactions which operate upon the schema. A distinguishing feature of this system is that the user is not required to reference the schema in the definition of the transactions.

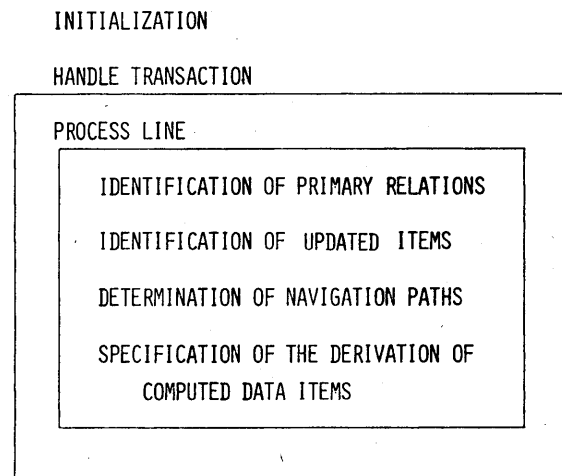


Figure 7—General organization of write transactions

### Further work

At present the system's final output is a set of transactions in a modified form of SEQUEL2. Extensions are planned to incorporate maintenance utilities into the system. The addition of these features would permit the user to construct an entire application system.

As mentioned in the body of this paper, certain limitations presently exist in terms of the operations available within the transactions that may be defined. These restrictions are a reflection of the prototype status of the system and are expected to disappear as further work on the system transpires.

An important feature of any system with a high degree of interactivity is the understandability of the questions by the user. An effort has been made to involve representative users in the design of the interactive phases of the system. It is hoped that the effect of this involvement will be easy understanding of the system by the casual user. If the user can interact easily with the system, then the database and transactions can be designed without requiring special training.

### REFERENCES

1. Beeri, C. and Bernstein, P. A., "Computational Problems Related to the Design of Normal Form Relational Schemas," *ACM TODS* (4, 1), Mar. 1979, pp. 30-59.
2. Bernstein, P. A., "Synthesizing Third Normal Form Relations from Functional Dependencies," *ACM TODS* (1, 4), Dec. 1976, pp. 277-298.
3. Buser, R. H. and Kusnyer, S. K., "Automatic Synthesis of Third Normal Form Relations," M. S. Report, Comp. Sci. Dept., Kansas State U., Dec. 1977.
4. Chamberlin, D. D., "Relational Data-Base Management Systems," *Computing Surveys* (8, 1), Mar. 1976, pp. 43-66.
5. Chamberlin, D. D. et al., "SEQUEL2: A Unified Approach to Data Definition, Manipulation, and Control," *IBM Journal R&D* (20, 6), Nov. 1976.
6. Chang, S. K. and Ke, J. S., "Translation of Fuzzy Queries for a Relational Database System," Knowledge Systems Lab., U. of Illinois at Chicago Circle, Mar. 1978.
7. Christensen, M. A. and Herndon, M. A., "QUEASY: The Design and Implementation of a Management Information System for Casual Users," *ACM Annual Conf.*, Dec. 1978, pp. 230-233.
8. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *CACM* (13, 6), June 1970, pp. 377-387.
9. Codd, E. F., "Seven Steps to Rendezvous with the Casual User," in *Data Base Management*, (J. W. Klimbie and K. L. Koffeman, eds.), North-Holland, Apr. 1974, pp. 179-200.
10. Codd, E. F., "How About Recently," in *Databases: Improving Usability and Responsiveness* (B. Shneiderman, ed.), Academic Press, 1978, pp. 3-28.
11. Greenblatt, D. and Waxman, J., "A Study of Three Database Query Languages," in *Databases: Improving Usability and Responsiveness* (B. Shneiderman, ed.), Academic Press, 1978, pp. 77-97.
12. Harris, L. R., "The ROBOT System: Natural Language Processing Applied to Data Base Query," *ACM Annual Conf.*, Dec. 1978, pp. 165-172.
13. Hendrix, G. G. et al., "Developing a Natural Language Interface to Complex Data," *ACM TODS* (3, 2), June 1978, pp. 105-147.
14. Hunt, W. O., "Interactive Generation of Functional Dependencies," M. S. Report, Comp. Sci. Dept., Kansas State U., Dec. 1977.
15. Lozinskii, E. L., "Performance Considerations in Relational Data Base Design," in *Databases: Improving Usability and Responsiveness* (B. Shneiderman, ed.), Academic Press, 1978, pp. 273-294.
16. Powell, P. B. and Thompson, P., "Natural Language and Voice Output for Relational Data Base Systems," *ACM Annual Conf.*, Dec. 1978, pp. 585-595.
17. Reischer, P., "Use of Psychological Experimentation as an Aid to Development of a Query Language," *IEEE Trans. on Soft. Eng.* (SE-3, 3), May 1977, pp. 218-229.
18. Roush, C. S., "A User-oriented Transaction Definition Facility For a Relational Database System," M. S. Report, Comp. Sci. Dept., Kansas State U., Aug. 1979.
19. Shen, S. N. T., "A Semantic Approach in Designing Relational Data Base," *ACM Annual Conf.*, Dec. 1978, pp. 596-601.
20. Shneiderman, B., "Improving the Human Factors Aspect of Database Interactions," *ACM TODS* (3, 4), Dec. 1978, pp. 417-439.
21. Sorenson, D. G. and Wald, J. A., "PICASSO-An Aid to an End-User Facility," *ACM SIGMOD Conf.*, Aug. 1977, pp. 30-39.
22. Stevens, T. J., "Implementation of a Text Editor for the Third Normal Form Synthesis System," M. S. Report, Comp. Sci. Dept., Kansas State U., Dec. 1977.
23. Waltz, D. L., "An English Language Question Answering System for a Large Relational Database," *CACM* (21, 7), July 1978, pp. 526-539.
24. Weiner, J. L., "Deriving Data Base Specifications from User Queries," *Berkeley Workshop on Distributed Data Management and Computer Networks*, May 1977, pp. 182-195.
25. Zloof, M. M., "Query-By-Example," *National Computer Conf.*, Vol. 44, May 1975, pp. 431-437.
26. Zloof, M. M. and de Jong, S. P., "The System for Business Automation (SBA): Programming Language," *CACM* (20, 6), June 1977, pp. 385-396.

### APPENDIX

#### 1. Document Descriptor to STUDRECD Document

Name	Level	Group	Flag
STUD#	1	110	1
STUDNAME	1	110	0
GRADDATE	1	110	0
GPA	1	110	0
STUD#	2	210	1
COURSEWORK	2	210	1
COURSEGRADE	2	210	0

## 2. Relations

R01 STUD# STUDNAME GRADDATE GPA  
 R02 STUD# COURSEWORK COURSGRADE

## 3. User Dialog with Transaction Definition Subsystem

TDS:           STARTING NEW TRANSACTION           (REPORT)  
 TDS:                   TRANSACTION-NAME:           STUDRECD  
 TDS:       STUD#           STUDNAME           GRADDATE           GPA  
 TDS: WHICH DO YOU WANT TO DISPLAY  
 TDS:           Y—YES, THIS ITEM  
 TDS:           N—NOT, THIS ITEM  
 TDS:           A—ALL OF THE REST OF THE ITEMS  
 TDS:           S—STOP, NONE OF THE REST OF THE ITEMS  
 TDS:   STUD#  
 USER:   Y  
 TDS:   STUDNAME  
 USER:   Y  
 TDS:   GRADDATE  
 USER:   Y  
 TDS:   GPA  
 USER:   Y  
 TDS: DO YOU WANT THIS TRANSACTION TO INVOLVE EVERY  
 TDS: OCCURRENCE OF THE DATA FOR THIS LINE? (Y/N)  
 TDS: ('N' IF THERE IS ADDITIONAL SELECTION CRITERIA)  
 USER:   N  
 TDS: OF THESE ELEMENTS IN THE LINE:  
 TDS:   STUD#           STUDNAME           GRADDATE           GPA  
 TDS: WHICH IS INVOLVED IN THIS CONDITION?  
 USER:   STUD#  
 TDS: PLEASE ENTER CONDITION OPERATOR (=, <, >, ETC.)  
 USER:   =  
 TDS: FOR NOW, THE 'RIGHT-HAND SIDE' WILL COME FROM  
 TDS: THE CRT  
 TDS: OF THESE ELEMENTS IN THE LINE:  
 TDS:   STUD#           STUDNAME           GRADDATE           GPA  
 TDS: WHICH IS INVOLVED IN THIS CONDITION?  
 USER:   STUD#           STUDNAME           GRADDATE           GPA  
 TDS: DO YOU WANT THIS LINE SORTED? (Y/N)  
 USER:   Y  
 TDS: ENTER SORT KEYS MAJOR TO MINOR  
 TDS:   STUD#  
 USER:   Y  
 TDS: ASCENDING OR DESCENDING? (A/D)  
 USER:   A  
 TDS: ARE THERE ANY MORE SORT KEYS? (Y/N)  
 USER:   N  
 TDS:   STUD#           COURSEWORK           COURSGRADE  
 TDS: WHICH DO YOU WANT TO DISPLAY  
 TDS:           Y—YES, THIS ITEM  
 TDS:           N—NOT, THIS ITEM  
 TDS:           A—ALL OF THE REST OF THE ITEMS  
 TDS:           S—STOP, NONE OF THE REST OF THE ITEMS  
 TDS:   STUD#  
 USER:   N  
 TDS:   COURSEWORK

```

USER:    Y
TDS:    DO YOU WANT THIS TRANSACTION TO INVOLVE EVERY
TDS:    OCCURRENCE OF THE DATA FOR THIS LINE? (Y/N)
TDS:    ('N' IF THERE ARE ADDITIONAL SELECTION CRITERIA)
USER:    N
TDS:    OF THESE ELEMENTS IN THE LINE:
TDS:    STUD#      COURSWORK      COURSGRADE
TDS:    WHICH IS INVOLVED IN THIS CONDITION?
USER:    STUD#
TDS:    PLEASE ENTER CONDITION OPERATOR (=, <, >, ETC)
USER:    =
TDS:    FOR NOW, THE 'RIGHT HAND SIDE' WILL COME FROM
TDS:    THE CRT
TDS:    OF THESE ELEMENTS IN THE LINE:
TDS:    STUD#      COURSWORK      COURSGRADE
TDS:    WHICH IS INVOLVED IN THIS CONDITION?
USER:    COURSWORK
TDS:    PLEASE ENTER CONDITION OPERATOR (=, <, >, ETC)
USER:    =
TDS:    FOR NOW, THE 'RIGHT HAND SIDE' WILL COME FROM
TDS:    THE CRT
TDS:    OF THESE ELEMENTS IN THE LINE:
TDS:    STUD#      COURSWORK      COURSGRADE
TDS:    WHICH IS INVOLVED IN THIS CONDITION?
USER:
TDS:    COURSWORK      COURSGRADE
TDS:    DO YOU WANT THIS LINE SORTED? (Y/N)
USER:    Y
TDS:    ENTER SORT KEYS MAJOR TO MINOR
TDS:    COURSWORK
USER:    Y
TDS:    ASCENDING OR DESCENDING? (A/D)
USER:    A
TDS:    ARE THERE ANY MORE SORT KEYS? (Y/N)
USER:    N
TDS:    DO YOU WANT TO DISPLAY ALL VALID RECORDS,
TDS:    OR JUST THOSE THAT ARE UNIQUE? (A/U)
USER:    A

```

#### 4. Output

```

TRANSACTION-NAME      STUD-RECD
ACCEPT PARAMETER-01  FROM CRT
SELECT (NEXT)
  R01.STUD
  R01.STUDNAME
  R01.GRADDATE
  R01.GPA
FROM
  R01
WHERE
  R01.STUD# = PARAMETER-01
ORDERED BY
  R01.STUD#  ASCENDING

ACCEPT PARAMETER-01 FROM CRT
ACCEPT PARAMETER-02 FROM CRT

```



```
SELECT (ALL)
  R02.COURSEWORK
  R02.COURSEGRADE
FROM
  R02
WHERE
  R02.STUD# = PARAMETER-01
  AND   R02.COURSEWORK = PARAMETER-02
ORDERED BY
  R02.COURSEWORK  ASCENDING
```

# Programming with data frames for everyday data items\*

by DAVID W. EMBLEY

University of Nebraska  
Lincoln, Nebraska

## INTRODUCTION

Processing everyday data items such as dollar amounts, time, dates, and account numbers constitutes a significant portion of real-world computer applications. Programmers involved with everyday data items confront the drudgery of writing routines to recognize, validate, transform, store, retrieve, manipulate, and display these items and also the challenge to develop user-friendly data-entry systems and insure data integrity. They usually meet these challenges using various and sundry ad hoc techniques.

Sometimes, much of the burden is transferred to data-entry personnel who are asked to adhere to rigid input formats and to insure accuracy by tedious double checking. As explained by Gilb and Weinberg in their book *Humanized Input*, many poor system designs have been saved by the accurate touch of the keypunch operators.<sup>1</sup>

In an attempt to systematically address the problems encountered when processing everyday data items, the concept of a *data frame* is proposed. A data frame provides a means to encapsulate the concept of a data item with all of its essential properties including alternative natural language written forms, computer representation, applicable contextual information, permissible operations, and relationships with other data items.

## DATA FRAMES

The name "data frame" has been coined because of the concept's similarity to data abstractions<sup>2,3</sup> and Minsky frames.<sup>4</sup> A data frame can be thought of as an extension to data abstractions or as a Minsky frame cast in the form of an abstract data type.

Minsky's theory of frames is a theory of rich symbolic structure where a frame represents a particular situation. Included in the frame is information about how to use the knowledge, what can be expected, and what to do if expectations are not confirmed. Data frames represent data items instead of situations, but the information included and its purpose are quite similar.

An attempt to precisely define the syntactic structure that includes all information that might be needed for all possible applications is premature, but it seems reasonable to extend the structure of data abstractions to represent the additional information required. The aim is to appropriately model the behavior of a particular data item.

Figure 1 shows some of the essential features for a data frame for dollar amounts. There are several acceptable written forms for U. S. currency, for example, \$25.63, \$2,638,457.00, \$.63, 63¢, \$47, 47\$ or just plain 25.63 when the context is understood. In general, the dollar amount input routine should accept any of these forms and perform the necessary translation to an internal computer representation. A return code supplied by the input routine gives the completion status and provides information for error messages. The corresponding output routine produces a formatted string ready to be displayed. A more detailed description of input/output conversions is given elsewhere.<sup>5</sup>

The context keywords in Figure 1 are extracted from the forms in common use in the Computer Science Department at the University of Nebraska. In the context of one of these keywords, if a data item is expected, it is quite certain that the type of the data item is a dollar amount. Also extracted from the forms are the operations addition, subtraction, and multiplication by an integer, along with context keywords that indicate an operator's applicability.

A library of commonly used data frames would be a valuable asset to application programmers who could then extract, possibly modify, and use them along with data frames created by themselves to fit their needs. A general dollar amount data frame taken from a library, for instance, could be adopted for use in an application involving UNL Computer Science departmental forms with very little if any alteration. The general library version might have a somewhat longer or different list of context keywords and an additional operator or two, but would be essentially the same.

Data frames can also be grouped together to model items more complex than a data element. Indeed, a data frame group is indistinguishable from a data frame for an elementary item except that selector operations would be available to provide access to an item within the group. A date, for instance, may be constructed as an elementary data frame or as a data frame group with selector functions for day, month, and year.

\* This material is based upon work supported by the National Science Foundation under Grant No. MCS-7904126.

```

data frame dollar_amount;

internal representation real;

input (s: string) returns (dollar_amount, return_code);
  (* validate string s and translate it into the internal representation *)
  ...
  end;

output (a: dollar_amount, f: format specification) returns (string, return_code);
  (* translate a dollar amount variable a into a string according to a
    given format specification f *)
  ...
  end;

context keywords: amount, amt, budget, cost, dollar, expense, extension, funds,
  honorarium, income, price, rate, sal, salary, sales, total, $;

infix function (a1: dollar_amount) + (a2: dollar_amount)
  returns (dollar_amount);
  context keywords: add, total, +;
  return (a1+a2);
  end;

infix function (a1: dollar_amount) - (a2: dollar_amount)
  returns (dollar_amount);
  context keywords: difference, subtract, -;
  return (a1-a2);
  end;

infix function (a: dollar_amount) * (n: integer) returns (dollar_amount);
  context keywords: amount, extension;
  return (a*n);
  end;

end.

```

Figure 1—Illustration of some of the essential features for a dollar amount data frame

In the next three sections, examples are presented to show how data frame features might be utilized.

#### *Computer-assisted instruction example*

One of the problems with CAI is the lack of flexibility in accepting student responses. The main reason for this rigidity is that input validation and interpretation is so difficult. For questions that require common items of data as answers, data frames can help make the development of user-friendly input requirements easier.

Figure 2 shows several student responses to the question, "When was Abraham Lincoln born?", and a reasonable system reply to each response based on information from a data frame for dates. A date input routine would reject "Kentucky" and "32 Feb, 1890" as invalid dates. Both "2/12/1809" and "12/2/1809" would be acceptable because the ambiguity could be resolved by comparing with the expected answer. By making use of the available date operations, dates that are too early, late, or only partially correct could also be identified.

When was Abraham Lincoln born?

```

> Kentucky
    No, the answer should be a valid date.
    Try again.
> 32 Feb 1809
    No, the answer should be a valid date.
    Try again.
> 2/12/1809
    Ok, your answer is correct. (February 12, 1809)
> 12/2/1809
    Ok, your answer is correct. (February 12, 1809)
> November 17, 1805
    No, he was born after that.
    Try again.
> Jan. 4, 1815
    No, he was born before that.
    Try again.
> 12-2-1808
    No, the year is wrong.
    Try again.
> 12 February 1809
    Ok, your answer is correct.

```

Figure 2—Student responses (preceded by ">") and CAI-system replies to a question requiring a date for an answer

### Query language example

At the University of Nebraska, several students and faculty in the Department of Computer Science together with personnel from the Computing Network are in the process of developing SIMPLE, a programming environment designed for instructional use for beginning students and students with minimal system requirements.<sup>6</sup> As part of the project, a database has been established containing information about student users such as account number, name, account balance, processor usage, and available command set. A query language was hastily provided to enable instructors to access and update the information, but a more user-friendly interface based on data-frame ideas is being designed.

The data-frames query language has no syntactic structure, neither is it based on natural language processing concepts. Instead, self-identifying data in its natural language form, context keywords, and available operators provide the information necessary for inferring the desired action.

Figure 3 shows a sample dialogue between a class instructor and the system. In the first request, the "\$25" can be recognized as a dollar amount; then, with the context keyword "+" and the lack of any account designation, the sys-

tem proposes that all accounts receive an additional 25 dollars. In the second request, the English phrase should not be construed to carry meaning; the presence of the keyword "ADD" along with a dollar amount and account numbers is all that is necessary. In the third request, a context dependency arises since the instructor has just singled out two specific accounts. In a second attempt, the discrepancy is resolved. For this request, of course, a data frame on SIMPLE commands would have to be available to recognize GET, PUT, and APPEND as commands and to recognize ADD as a context keyword associated with set union. (SIMPLE allows an instructor to tailor its use by specifying which commands are available to each user.) The only reasonable response to a name when no context information applies is to display information about the individual, and since the database is not being altered, no confirmation is necessary. (In SIMPLE all command keywords are uniquely identified by their first letter.) In the fifth request, SINCE is not recognized, so more than one reasonable alternative exists. If an impasse is reached where the system sees no other reasonable alternatives, the query language provides a structured request format that can be used as a last resort.

### Forms-based programming example

In a forms-based approach to programming, an application programmer describes data processing operations by means of structures modeled on conventional administrative forms. For a particular function, the programmer designs the form, specifies the data type for each item on the form, and defines relationships among the items. A system of related forms can be defined so that information is manipulated and routed among forms, a database, and external devices.<sup>7</sup>

```

>+$25
ADD $25 TO ALL ACCOUNTS? (Y/N)
>Y
DONE

>ADD AN ADDITIONAL $10 TO ACCOUNTS CSC0321 AND CSC0607.
ADD $25 TO CSC0321 CSC0607? (Y/N)
>Y
DONE

>ADD GET, PUT, AND APPEND
INCLUDE GET PUT APPEND IN COMMAND SET FOR CSC0321 CSC0607? (Y/N)
>N
INCLUDE GET PUT APPEND IN COMMAND SET FOR ALL ACCOUNTS? (Y/N)
>Y
DONE

>MARK MEYER
ACCOUNT # NAME BALANCE NR. RUNS COMMANDS
CSC0097 MARK MEYER $43.17 12 ABCDFGHILNOPQRT

>NUMBER OF PASCAL RUNS FOR ALL USERS SINCE 12 APR
ON APRIL 12, 1979? (Y/N)
>N
FROM APRIL 12, 1979 TO PRESENT? (Y/N)
>Y
493

```

Figure 3—SIMPLE query language (lines preceded by ">" are input by the user)

Suppose that the invoice-voucher form shown in Figure 4 is part of a system of forms that models a business operation. As an application programmer designs the form, blank spaces for expected entries are specified to be of a certain type and are associated with key phrases that indicate what entries are expected. The blank spaces for vendor name, street, city, state, zip code, and voucher total \$ all expect a single entry. Item no., description, quantity, unit price, and amount are members of a group of items each containing an unknown but equal number of entries.

With a library of appropriate data frames at its disposal, it is conceivable that a forms-based programming system could select the appropriate frame for each item on the form and define relationships among the items without any intervention from an application programmer. For the dollar amount entries, the associated key phrases all contain context keywords that appear in the data frame in Figure 1. It is not hard to imagine that the other needed data frames

likewise contain the essential context keywords to enable the entry type to be determined.

In addition to context keywords, data frames used for forms-based programming also require knowledge about the nature and layout of forms. The addition operator in the dollar amount data frame, for instance, must know that a column of dollar amounts can be added together to produce a total. For the invoice-voucher, this information coupled with the expectation of a total in the blank space immediately below the column of dollar amounts is enough for the system to propose the relationship  $\sum \text{amount}_i = \text{voucher total } \$$ .

Rows also imply relationships. Record information is often placed on a row; and therefore, if the item no., description, and unit price match field descriptors for records in a database file, values for these entries could be either double checked against a database or partially acquired from the file given only the item no.

The relationship  $\text{quantity}_i * \text{unit price}_i = \text{amount}_i$  is also de-

INVOICE-VOUCHER

P	vendor name
A	street
E	city
E	state
	zip

item no.	description	quantity	unit price	amount
				voucher total \$

Figure 4—Invoice-voucher form

ducible from the facts at hand. Where relationships cannot be deduced or where ambiguities or misunderstandings arise, an application programmer can supply the information or resolve the issue and always has the final say.

#### CONCLUDING REMARK

Although much remains to be done, even a step toward the encapsulation of the essential properties of everyday data items in data frames would benefit programmers in many applications. Extending data abstractions with input/output routines alone could simplify many programming tasks particularly for business and interactive computing applications where handling input and output constitutes a significant portion of the programming effort.

#### REFERENCES

1. Gilb, T. and Weinberg, G. M., *Humanized Input*, Winthrop Publishers, 1977.
2. Guttag, J. V., Horowitz, E., and Musser, D. R., "The Design of Data Type Specifications," in *Current Trends in Programming Methodology*, R. T. Yeh (ed.), Prentice-Hall, 1978, pp. 60-79.
3. Liskov, B. H. and Zilles, S. N., "Programming with Abstract Data Types," *Proceedings of ACM Symposium on Very High Level Languages, SIG-PLAN Notices*, Vol. 9, No. 4, April 1974, pp. 50-59.
4. Minsky, M., "A Framework for Representing Knowledge," in *The Psychology of Computer Vision*, P. H. Winston (ed.), McGraw-Hill, 1975, pp. 211-277.
5. Embley, D. W., *Data Abstractions for Everyday Data Items*, Department of Computer Science, University of Nebraska-Lincoln, August 1979.
6. Embley, D. W. and Nagy, G., *SIMPLE Specifications*, Department of Computer Science, University of Nebraska-Lincoln, June, 1979.
7. Embley, D. W., "Forms-Based Automatic Program Generation," *ACM 78 Proceedings*, December 1978, pp. 972-979.



# Implementing data management

by DANIEL S. APPLETON

*Appleton and Associates*  
Manhattan Beach, California

"If the organization plans to use database as a technique and a discipline for the long-term development of integrated applications, then there is far more to think about than if database is to be used simply as a sophisticated access method for the development of single, complex applications."

A database administrator  
Infosystems 10-10-79

Most businesses go through three distinct phases as they implement computer technology. In the first phase, they automate specific, reasonably well defined processes such as payables, receivables, invoicing and sales. In phase two, old applications are integrated and new applications are added either as integrated processes or as stand alone applications. In the third phase, a company database is established and users are given access to the information stored there.

According to most experts, today, only a few businesses are in phase three, though most are headed there. The vast majority of businesses are in the latter stages of phase one or in the initial stages of phase two. In short, the momentum toward database technology and data management strategies is rapidly mounting.

One of the basic reasons for this movement toward phase three is a fundamental change in how business sees the computer. Phase one companies generally perceive the computer as a device to control certain business processes. Phase three businesses treat the computer as a source of information which can be used to both control business processes and to help make decisions. In phase three companies, computer hardware, i.e., the mainframe or the mini-computer, becomes a means to an end—not an end in itself. Its purpose is to support the company database(s).

The general impression is that the movement from an applications-oriented DP management approach (phase one) to a data-oriented DP management approach (phase three) is more evolutionary than it is revolutionary. This is undoubtedly why very few companies achieve phase three. By treating the migration toward data management as a casual process, companies tend to get bogged down as they enter phase two. They end up only partially implementing data management. This they achieve by token changes in the tools they use (e.g., a database management system, a data dic-

tionary or some "user friendly" programming language) or the structure they employ (e.g., database administration, programmer teams). Gradual implementation of changes in tools and structures tends to result in poor implementation of data management. This can be read as phase one management using phase three tools and structures. It should not be misinterpreted as phase three.

The movement to data management should be treated as revolutionary rather than evolutionary. This does not mean it should be done overnight, in great haste or with massive personnel changes or infusions of money. It does mean that data management should be understood to mean changes in the total approach to automation. Not only will tools change—requiring significant investments, retraining and software conversions—but structures will change as will methods of doing things. Structured changes, as we shall see, will affect more than just DP—the user will also be affected. And, as far as methods, they should be the first to change. Among them, probably the most important method to change is the approach used for designing and implementing software. This method is often called software engineering. The revolution which will result in phase three must be felt in all three of the major dimensions of the problem: DP tools, structures and methods, especially in software engineering.

## AT THE HEART OF DATA MANAGEMENT IS DATABASE

Before examining each of the three dimensions of the data management problem, we must examine data management itself. The roots of data management are buried deep in the concept we know as *database*. In the early days, database was just a set of computer software called a Data Base Management System (DBMS). Its purpose was simply to maximize the accessibility of data stored in files on the computer. Subsequently, "database" developed a broader meaning. It came to be an approach to data processing management which rivaled the conventional approach of applications management. It is this concept of database which evolved to become "data management."

To fully comprehend what data management has come to mean in today's world of data processing, we must coin a new word: IRRASSPA, pronounced ear-ē-asp'-ā. This word is actually an acronym for the nine major attributes of



data management. Each letter, as we shall see, stands for something that is noticeably deficient in the conventional data processing environment.

The I in IRRIASSPA stands for data "independence." Very simply this means that data stored on the computer must be independent from the programs which access and manipulate it.

The R stands for data "reliability." A little more difficult to understand, this means that the structure and relationships among the data stored in the database must be flexible at all times. Thus we must be able to change the content of the database not just in terms of data stored but also in terms of the relationships among those data.

The second R stands for data "non-redundancy." This means that we should be able to store data only once and still have it available for use by all computer programs.

The I in IRRIASSPA stands for data "integrity." Maintaining the integrity of data stored on the database is a crucial function of the data management environment.

The A stands for data "accessibility." Data stored on the database must be accessible in an infinite variety of ways, through cathode ray tubes, hard copy reports, graphic displays, microfilm, etc.

The first S stands for data "shareability." This means that different computer programs written in different languages should be able to share a common database. Also, they should be able to do so without each having to be aware that the other exists.

The second S in IRRIASSPA stands for data "security." The ability to secure data on the database in order to assure against malicious actions is crucial to a good data management environment. Data security is necessary on both the input and the output sides of the database.

The P in IRRIASSPA stands for "performance." Data processing departments must have the capability to control processing performance of the database as the demands against it change through the years. They must be able to restructure the database and modify it based upon performance characteristics and upon the demands of individual users.

And finally, the A stands for database "administration." Very simply, this means that the database must be managed as a company asset. This in itself is a new concept. Traditionally computer applications systems are managed as departmental expense items. But if a database is created to support the whole business, then that database must be viewed in much the same manner as the other essential elements of the businesses asset structure.

IRRIASSPA is data management. Without IRRIASSPA, data management is unattainable. But, IRRIASSPA is not, as many seem to believe, just a DBMS. Nor is it a database administrator. It is much more.

To establish a data management environment, data processors must change their whole approach to data processing. They must change their structure, their tools and their approach to software engineering. Each of these is a crucial dimension of the data management issue (see Figure 1).

First and foremost, DP must overcome the tremendous inertia behind the traditional, deterministic approach to software engineering. This traditional approach *will not work* in a data management environment. What is worse, it inhibits the whole development of data management. Software engineering is the bedrock of data processing. It is the take-away It sets the whole pace. If the methodology is not geared to produce software which is sympatico with data management strategies (IRRIASSPA), the result will be highly dissatisfying.

In addition to software engineering, the whole *structure* of data processing must be made compatible with data management. This will affect both DP and the user, and it will affect organization structures, job descriptions, organizational policies and operational procedures.

The final piece to the data management puzzle concerns the tools necessary to attain and retain control over IRRIASSPA. These tools involve both hardware and software, and more and more they involve capabilities in the area of communications. These tools, along with the appropriate database software engineering methodology and a well designed data management structure, will help data processing departments to attain the real objectives of data management: IRRIASSPA.

Failure to recognize and deal with each of the three dimensions of data management is the single most serious impediment to its implementation. DP managers who expect to gain its benefits while only doing half of the job are only fooling themselves; they are also fooling their employees, their users and their management.

## DATABASE SOFTWARE ENGINEERING

Marshall McLuhan became famous for writing: "The Medium is the Message." This phrase is tremendously important to data management. Software engineering is the process of designing and building database applications. It is singularly important in determining the products which will have to be maintained within the data management environment. Thus, there are several critical aspects to this software engineering methodology which we must consider. One of the most important is that the software engineering methodology must fit snugly within the data management structure. As we shall see, this data management structure contains certain basic operational activities which are ongoing and which are important for both flexibility and control.

Another crucial aspect of database software engineering is that it must make maximum use of data management tools. DBMS indexing capabilities, non-procedural programming languages, data dictionaries and directories, interactive query facilities, data definition languages, on-line update facilities, etc., must be integral parts of the development process, not just afterthoughts of the system design.

The primary target of database software engineering is the database itself. The database has meaning beyond the reports which it generates. These reports are its "kinetic"

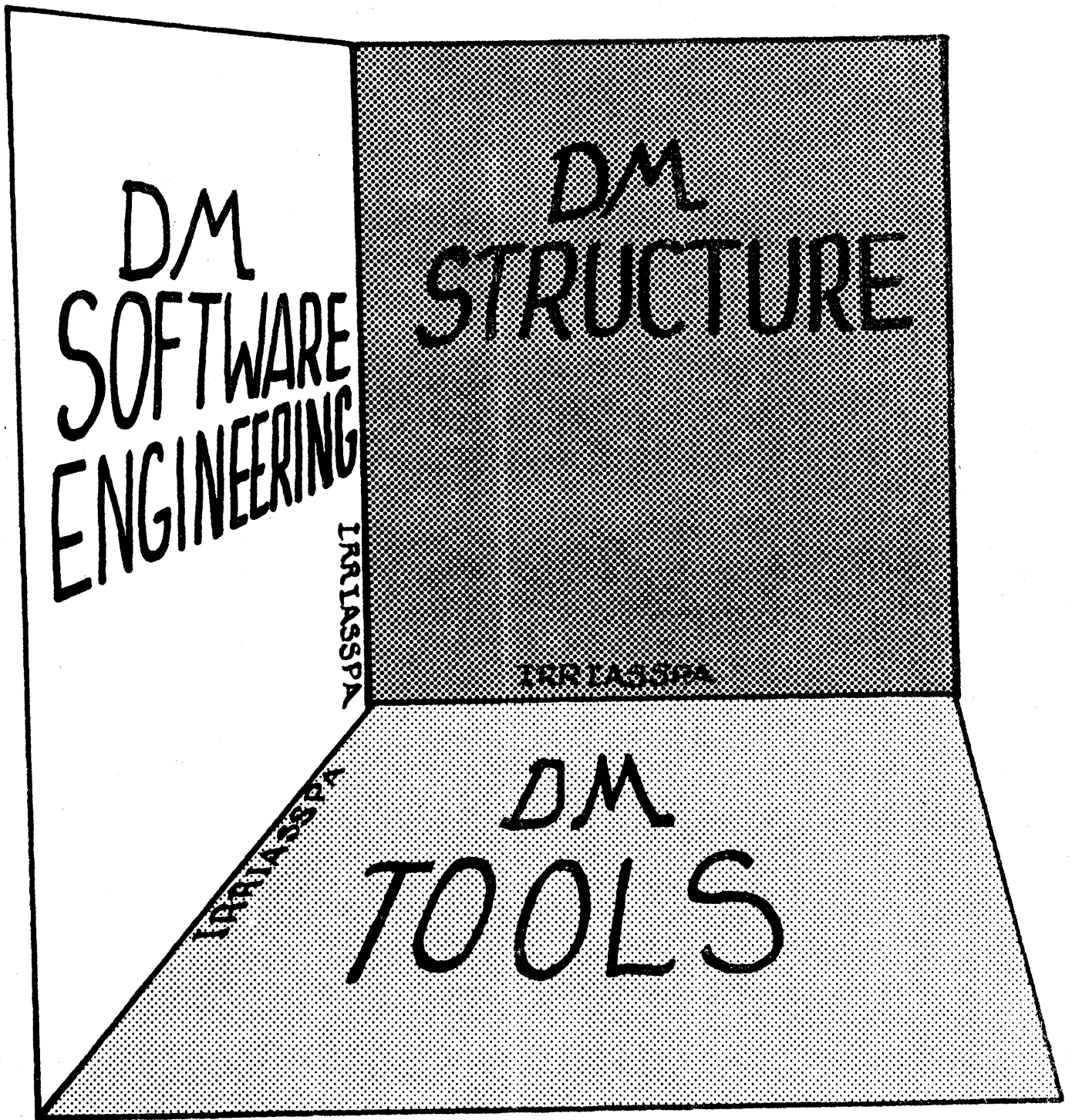


Figure 1—The three dimensions of data management.

energy. What conventional development methodologies ignore is the "potential" energy of the database. Database software engineering must maximize the database's potential by designing the database first, not last as is dictated by traditional structured design methodologies. Database soft-

ware engineering deals directly with data entities, attributes and relations. It does *not* deal obliquely with issues of database content and structure, looking at them through the haze of arbitrary, temporary output requirements. This is the conventional approach: freeze the user's reporting require-

ments, develop the input capabilities to be compatible with those requirements and, ultimately, design the data files to optimize the input to output conversion.

One of the major problems with the traditional approach to software development is that it purposefully excludes the user "experience" from the design process. This is why user involvement is always so low. (In most structured design methodologies it is less than 5 percent.) The user does not consider "involvement" to mean "report definition." The conventional development process does. To get the user involved, as he must be in a true data management environment, the development methodology must give him experience with the database so that he can understand its full capabilities and limitations. It must also allow him to change the database and to modify its applications (i.e., its outputs and its inputs) without necessarily resorting to complex procedural programming languages or data processing professionals for everything.

With objectives like designing the database first and integrating user experience in the design process, conventional software engineering methodologies quickly lose their appeal. They are too deterministic, and they depend too much on detailed design accuracy. What is more appropriate is a "heuristic" process, especially one which employs a "prototype." Such an approach used to be impractical, given state of the art of programming languages and systems analysis techniques. However, with today's capabilities in DBMS software and non-procedural programming languages, prototype development and heuristic analysis are practical, viable approaches to software engineering. But, they must be managed properly. They cannot be used casually.

Like conventional development techniques, database development techniques must be formal and rigorous. However, their approaches to software engineering are different in other important ways. Conventional development methodologies do not formally separate input technology and strategy from either output or database technology and strategy. This is a significant flaw, especially given the concepts of IRRASSPA. Nor do conventional methodologies allow for the use of non-procedural programming languages in the design stages. Instead, they concentrate on detailed design techniques to lay the foundations for writing structured procedural code. This is one key reason why conventional development methodologies entail so much rewrite (up to 25-45 percent, depending on the depth of the detailed design) and why they are so costly, inflexible and slow.

The last major consideration in database software engineering is that it must have a significant step which involves performance and tuning. This tuning concept must involve users so that they can see how the database reacts to various operational environments. Various statistical methodologies and computer software facilities can be used to build statistical analyses about the database structure, and these can in turn be used to make modifications to the database content and the relational structure of the database.

Everyone must agree that tuning is a critical aspect of data management environment. The difference between the da-

tabase software engineering approach and the conventional software engineering approach is that in database software engineering tuning takes place after the database is operational, while in the conventional software engineering methodology, it must take place in the initial design stages, i.e., before the code is laid. This is one of the main reasons why computer applications systems are so inefficient, and why so much time is spent in design and development. Design changes tend to reek havoc with efficiency.

The ideal database software engineering methodology probably contains six steps (see Figure 2). The first two steps are specifically geared to requirements analysis. The first is an Operational Audit specifically to reveal the "as is" situation and problems. The second step is Conceptual Design. This step defines the database scope and the "service functions" (applications) which the database will probably support. This Conceptual Design stage is not in infinite detail, but it should be fairly specific in terms of the data scope.

The third step of the database software engineering methodology is "Database Definition." Here the systems analysts and the users define the data content of the database, define the data structures, normalize the database and establish a *prototype*, actually installing the database on the computer. This last step is relatively easy to accomplish, especially with some of the new tools available to DP today.

The fourth step of the development process is "Heuristic Analysis." (Heuristic means "trial and error.") In Heuristic Analysis users should interact with the prototype database (on-line if possible) in order to finalize its structure and content. Also, in Heuristic Analysis it is necessary to build the input structures required to support the database.

The fifth step of the database software engineering methodology is to define both the standard and *ad hoc* reporting requirements of the database and to build and implement an output structure. We might call this step Environmental Test.

The final phase of development must be Performance Review. In this phase, DP must make evaluations necessary for final database tuning. This step closes the development loop because it can be directly related to the original step of Operational Audit.

## THE DATA MANAGEMENT STRUCTURE

The concept of a data management structure may be hard to grasp. DP managers are used to dealing with using organizations (composed of users) and DP departments (of analysts, programmers and the like). The traditional relationship is that between the "users" and the "used." Little has been done organizationally to change this relationship, though major changes are required for data management. These changes assault traditional DP concepts, and they affect our perceptions about things as fundamental to data processing as documentation techniques, programming strategies, systems responsibility structures, and system design content.

Data management is a much more participative environ-

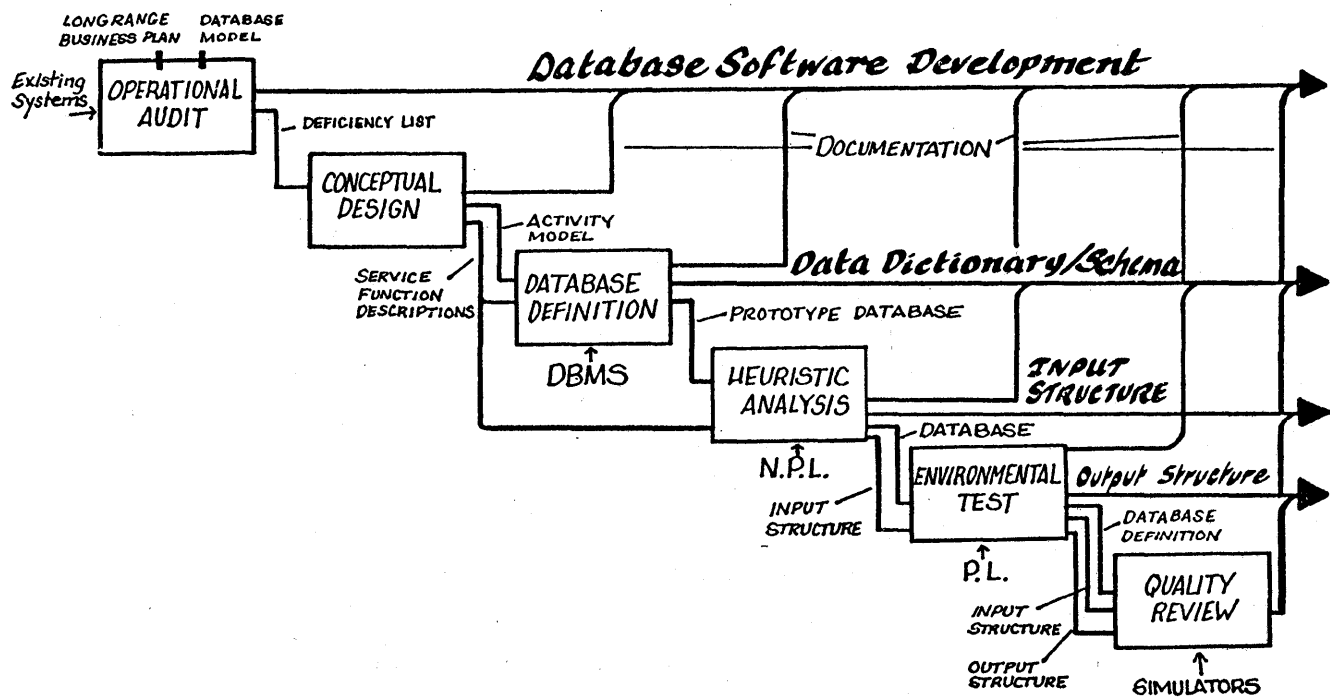


Figure 2—Prototype development methodology.

ment than is the applications environment. New institutions like database administration, input administration, output administration and information resource administration are inevitable. These new institutions are starting to replace the old "adversary" relationships created by conventional data processing management strategies, and they are beginning to fill in some of the gaps between "those that serve" and "those that are served."

Database administration is already accepted as an indispensable part of data management. However, there is no generally accepted definition of what a database administrator does; indeed, what he does today is to a large extent dependent upon his tools and his politics.

The population of database administrators is very small at this point in time, and the vast majority of those that are available have specialized in specific DBMS software. As a result, they tend to be more technical than not. This technical orientation of database administration is a reflection of its degree of maturity.

A closer look at the DBA function shows that while its description can range from very technical to very abstract, at its heart are four attributes of our data management environment: independence, shareability, reliability, and non-redundancy. If the DBA function does nothing else, it must insure that those four main objectives of IRRASSPA are in fact achieved. Of course, it would be natural to add "administration" to the functional responsibilities of database administration, thus giving it five major responsibilities with IRRASSPA. However in doing so, we expand its role beyond that of the technical management of databases into one which has more political overtones. In other words, we are

raising it up organizationally. It is of real concern whether any individual could competently handle such a broad spectrum of responsibilities.

Data management is more than just four or five of the aspects of IRRASSPA. It is all of them. Thus, we must either give our DBA function all of the responsibility for data management, or we must somehow supplement it. In other words, we must expand the data management structure beyond the DBA himself, or we must turn it all over to him. We have already seen that the latter may well be impractical. So, we must explore the former.

An examination of data processing from the functional perspective (i.e., database control, input and output) gives the clues needed for structuring data management. Figure 3 shows the resource requirements of these three critical DP functions. We can see that the demands of the output function are *exactly the reverse* of those for data storage and processing. Dynamics and responsiveness are low for the database while they are high for the output structure; quality requirements are relatively low for output while, along with technical expertise, they are high for data storage and processing. Input, as we can see, is also unique in its demands on resources. The lesson is that each of the three main DP functions places unique enough requirements to warrant its own administrative structure, with its own policies, procedures, job descriptions, budgets, objectives, technologies, etc.

Figure 4 is one possibility for this DP structure. In accordance with IRRASSPA, we can allocate I,R,R and S to the DBA. We can give the I of data Integrity to the input administrator. While we're at it, we might as well give the

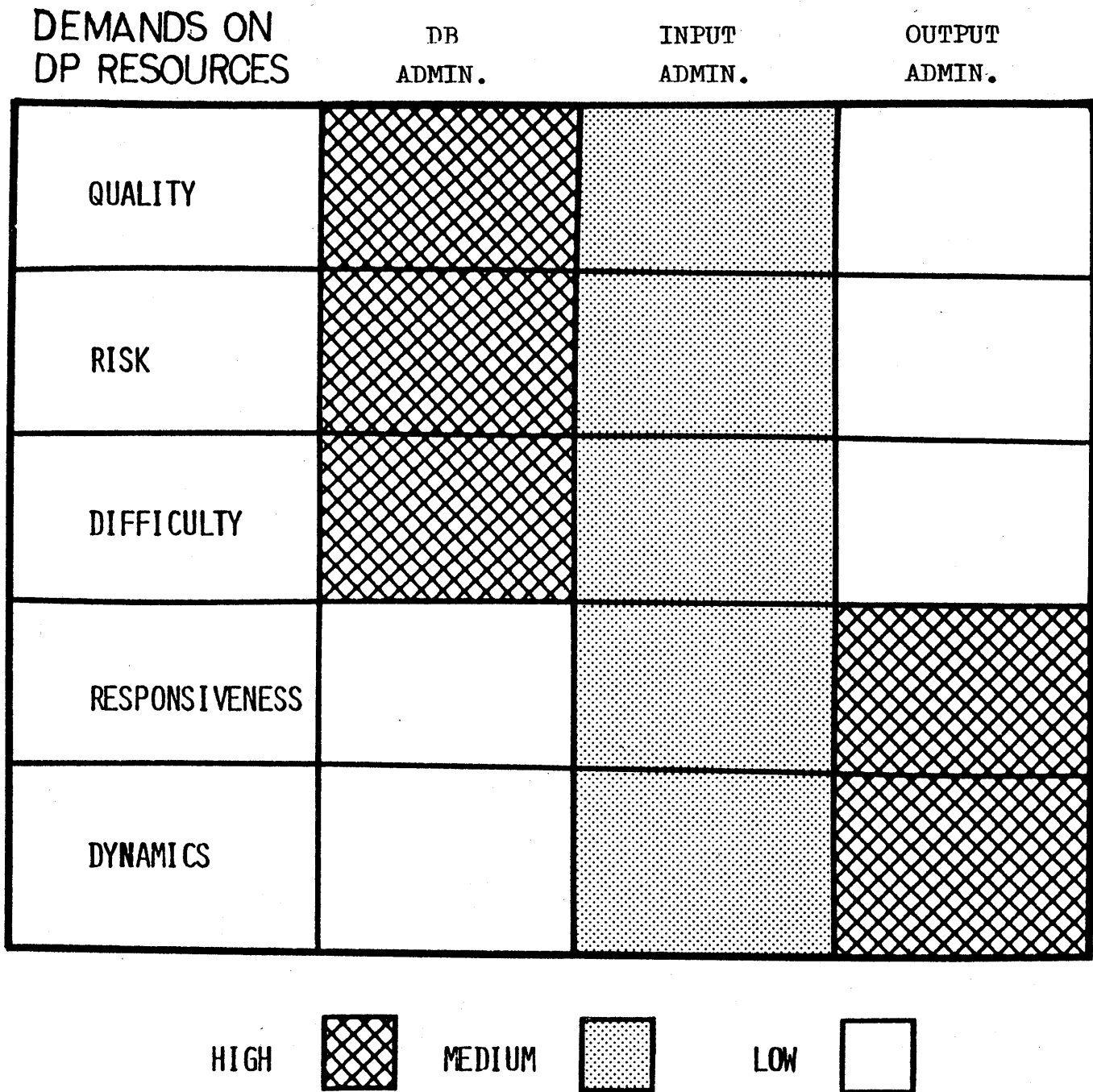


Figure 3—Demands on DP resources.

input administrator the programmers and the analysts which he will need to fill out his responsibility for input and update. The input administrator's responsibilities expand, however, beyond those simply required for data processing efficiency. He must also be concerned with functional procedures, e.g., input documents, input timing, development of input screens, input responsibilities, etc. He is required to maintain input integrity for the database, including checkpoint and restart facilities, on-line logging of up-dates, security, database re-loads and database restructuring.

The input administrator must also control production sequencing and the scheduling and updating, and he must have some responsibilities for database "S"ecurity as it applies to updating activities. Additionally, the input administrator has responsibility for edits, diagnostics, format checks and all other facilities necessary to maximize the quality of data in the database while minimizing the cost of getting it in there.

Since we have given the database administration functions and the input administration functions to various adminis-

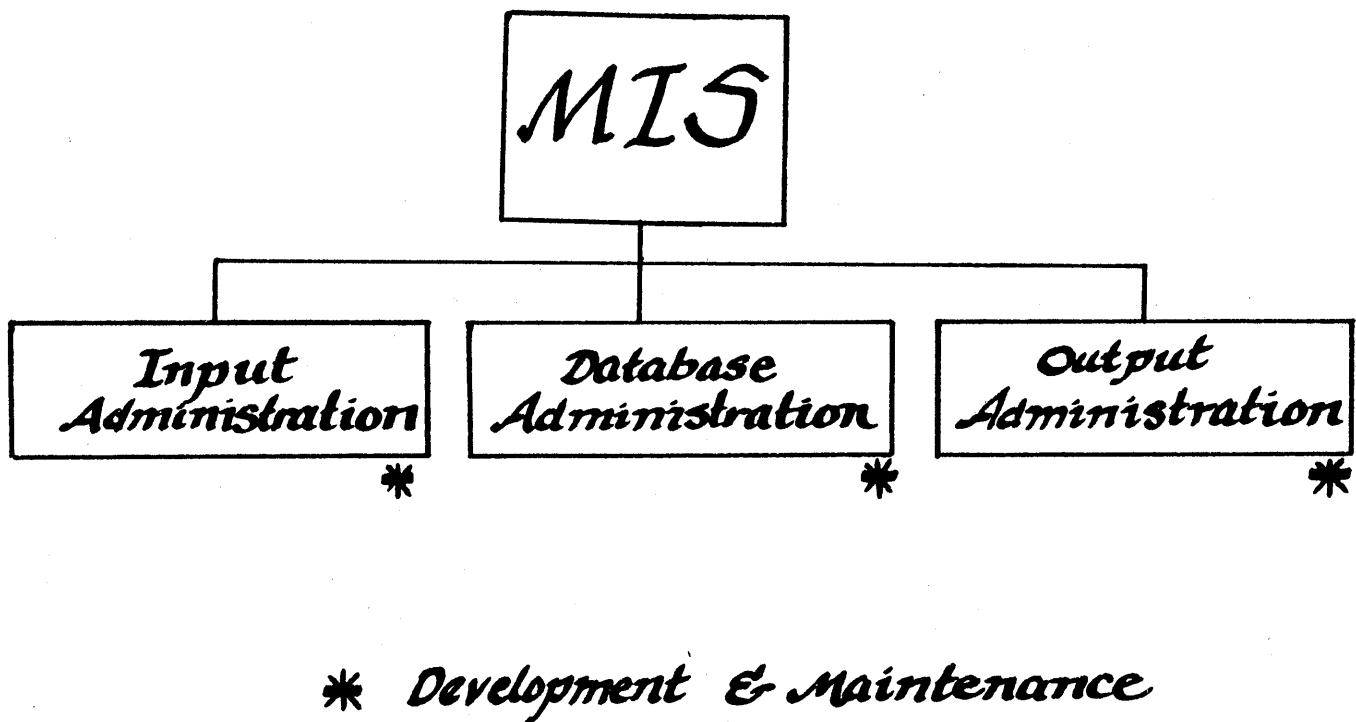


Figure 4—Data management structure.

trators, we are left with output. It seems only fair that the responsibilities for output would be given to an output administrator. The responsibilities of this function deal with the data "accessibility" portion of IRRASSPA. Like input administration function, these also deal with Security issues and with "P"erformance. The output administrator is responsible for selecting various query facilities for use against the database, for training users on retrieval techniques, for assuring production schedules, for developing reporting modes and media, and for generally assuring the proper alignment between the database and the user's requirements. The output administrator is mainly responsible to assure the database output is responsive to user needs at the lowest possible cost.

In addition to these three major administrative responsibilities within a data management environment, we must introduce a fourth: the Information Resource Administrator. There is usually one IRA for each major department head within the using organization. Being an IRA is not always a permanent, full-time job, so usually the functions of information resource administration are assigned to someone within a department; however, DP personnel could be distributed to user departments to perform the IRA role.

The responsibilities of the IRA are to monitor and document all departmental computer input and computer output, identify and define all departmental files, develop procedures and documentation, monitor the flow of work into and out of the department, and provide feedback to the database planning team about the efficiency of the department's information resources.

The IRA functions not only as a positive force in developing and implementing systems, he or she also has a significant role in maintaining systems. By calling all department IRA's together, a database planning team can do a simple audit of the efficiency of the flow of information throughout the company. This audit can be used to provide for not only corrective action, but also preventive maintenance on the part of user management and DP management.

IRA's are responsible for maintaining what is normally referred to as "user documentation." They are responsible for documenting all "discrepancies" to normal computer operations, and for officially requesting data processing services in support of their departmental needs, if those services are not directly related to their systems development plan. The latter recommendations should be approved by the database planning team before they are assigned budgets and schedules.

Another job of the Information Resource Administrator is to learn to program using simple "user friendly" languages and then to operate as a source of intra-departmental expertise for users who need to access the database in an *ad hoc* manner. He should do so without having to resort to the utilization of DP resources. The IRA is a focus for all data management training activities, and for participation on all new database development projects which affect his or her department. The IRA shares responsibilities with the other administrators for each and every component of IRRASSPA.

It is important to clearly establish the formal relationships among all four types of administrators. In doing so, the whole

DP/USER structure undergoes a metamorphosis. Users take on responsibilities they have never had in areas such as programming, documentation, maintenance, etc., and the data processors, while giving up some of their traditional prerogatives, undertake some new and more important ones.

There are other dimensions to the data management structure which, while less elemental, are still unique to data management. These affect planning, requirements analysis, database maintenance, etc. All of these functions look and act different in a data management environment. To leave them untouched is to leave the roots of future problems buried deep within the organization.

## DATA MANAGEMENT TOOLS

We come now to the third and, in many ways, the most familiar of the three dimensions of data management: data management tools. As little as five years ago, these tools were few and far between. Limited to so called Data Base Management Systems (DBMS), they addressed very few of the IRRASSPA attributes. Today, however, DBMS software is extremely elaborate; in fact, it is so elaborate that it is misleading to view it as just DBMS software. To be clear about its capabilities, we must look at its component parts.

For the sake of simplicity and understanding, we can develop six fundamental types of data management software tools. These include:

- Data Dictionaries/Directories (DD/D's)
- Database Definition Languages (DDL's)
- Database Manipulation Languages (DML's)
- Database Query Languages (DQL's)
- Database Control.
- Data Communications (DC) Facilities

Before briefly describing each type of software tool, I will try to erase any confusion which might exist between these facilities and what we conventionally refer to as a Data Base Management System (DBMS). Depending on the DBMS vendor, any given commercial DBMS may be composed of all six of these facilities, or only one or two of them. Usually, the more elaborate DBMS's have all of these facilities, but they are individually priced or priced in groups. Less elaborate DBMS's (such as those traditionally found on mini-computers) have only some of these facilities. Further, if compared facility by facility, some DBMS's are better in some than are their competitors and vice versa. I think it is a fair statement to say that no DBMS surpasses all of its competitors in all areas.

To effectively deal with IRRASSPA, all of these facilities are mandatory. Thus, to properly evaluate DBMS software tools, the DP department and the users should perform a careful analysis of their IRRASSPA needs. Only from this vantage point can they intelligently evaluate the various software offerings and select its software tools.

At first blush, DP management may feel that one vendor should provide all of these database software needs; however, if they assume that this is necessary, they are mistaken.

Further, they should not necessarily limit themselves to only one tool for each area. Multiple software tools, including multiple DDL's, DML's, DQL's, DD/D's, etc., are advisable under some conditions, depending, of course, upon data management's objectives, IRRASSPA strategies and company concerns.

Let's look more closely at each class of software tool.

### *Data dictionaries/directories (DD/D's)*

There are many approaches to DD/D's. Data dictionaries are most simply a list of all of the elements contained in the database and the relationships which are established among these elements. Usually a data dictionary describes each element within the database, its synonyms, the organization responsible for updating, any specific edits which are performed with regard to that data element, its security requirements and a description of what data element means. Data dictionaries also include the data element format as well as its character composition.

Usually supplementing the data dictionary is the data directory. The main job of the data directory is to describe how each individual data element is used, and where it is used. The data directory might point to various computer programs which use a data element; it might point to various job streams; or it might point to various input documents or computer reports. Comprehensive data directories might cover all of these areas in attempt to give a complete description of the utilization of every data element within the data dictionary.

In selecting a DD/D, it is important to determine whether it is integrated or not integrated with the DDL and the DML. It is also important whether the host DBMS DDL was used in constructing the DD/D. In the final analysis, however, the best DD/D's are those which the DP department feels it can maintain. DD/D's are not only difficult to establish; they are difficult to keep current—especially the directory features. An elaborate automated DD/D which can't be maintained is just as useless as is a manual one.

The DD/D is extremely critical in the data management environment. It has a direct affect on every aspect of IRRASSPA. In fact, in selecting the proper DD/D, data processing management must evaluate each of its IRRASSPA requirements and assure alignment between the capabilities of the DD/D and the requirements of each IRRASSPA component.

### *Data definition languages (DDL's)*

The term Data Definition Language was established by the Committee On Data Systems and Languages (CODASYL). It is used in conjunction with the concepts of database "schema" and "subschema." Basically, a schema is a description of a complete logical database, and a subschema is the description of a subset of that database which is utilized by an individual computer program. According to CODASYL, the main function of the DDL is to describe the content and structure of both the schema and the subschema.

All DBMS's have DDL's. Some of these DDL's are oriented toward describing data elements within hierarchies, and some of them are oriented toward describing networks. Some DDL's are much more elaborate than others, providing facilities for describing very complex relationships among data elements. Obviously, DDL's affect many of the aspects of IRRASSPA. For example, the quality of a DDL is in large measure derived by evaluating its ability to produce data Independence. Some DDL's are less efficient in establishing data independence than others. Some DDL's are more geared to reducing data Redundancy in describing certain types of database structures than are other DDL's. Another important feature of DDL's is whether or not they establish relationships among data elements with pointers embedded in the database or with pointers which are stored in indexes to the database. No DDL is best for all conditions. Some DDL's are easier to use than others, and some are more flexible. Selection of the proper DDL is of critical importance to data management. (No DDL is capable of describing a "structureless" database. This is the ultimate objective of the "relational" DDL. However, relational DDL's are not practical given the current state of hardware and software technology. Perhaps they will be in the future.)

#### *Data manipulation languages (DML's)*

Data Manipulation Languages (another CODASYL term) are used to *bind* together normal procedural languages such as COBOL and FORTRAN with the capabilities of the Data Definition Language. There are many different methods for binding the DDL to procedural languages, and therefore there are many different techniques that are used within Data Manipulation Languages. Most often, DML's are used by procedural language programmers as if they were *calls* to the DDL. These calls treat the DDL as if it were a set of sub-routines.

Facilities of the DML directly affect programmer productivity. Where the DDL is usually the province of the Database Administrator, the DML is usually the province of the procedural language programmer. The DDL is used by the DBA to provide subschema of the database for manipulation by the procedural language programmer with his COBOL or FORTRAN program. Thus, DDL's and DML's are often closely related. The functionality reserved for each usually describes the technical responsibilities which must be assumed by either the Database Administrator or the procedural language programmer. Some DML's require very little effort on the part of procedural language programmers. Other DML's force the programmer to write complex programs required to *navigate* through the database. Where a DML can require extensive navigation on the part of the procedural language programmer, a good DDL can act as his automatic pilot, thereby improving his productivity.

#### *Database query languages (DQL's)*

Database Query Languages come in many forms with many different capabilities. Most often, they are interactive,

on-line in nature; however, there are some very powerful "batch" DQL's. Database Definition Languages which store pointers in indexes as opposed to in the data file (often called "inverted list processors") often provide extremely powerful DQL facilities. These facilities allow direct interaction with the total database schema, and they allow parametric and Boolean search strategies to be used for either data retrieval or update.

DQL's are often referred to as "end user facilities." They are the attributes of the DBMS software which are said to make it "user friendly." Ultimately, DQL capabilities will turn over to users functions which traditionally have belonged to data processors.

While the DDL structure will remain under the control of the data processing department, the DQL structure will move under the control of using departments. The DML activities will, in all probability, remain twilight zone in nature, being shared by both data processors and users who are technically capable. In referring back to our data management structure, DBA's will use the DDL, Input Administrators and Output Administrators will primarily use the DML's, and the Information Resource Administrators will primarily use the DQL's.

#### *Database utilities*

Every data processing organization requires specific software utilities to improve its capability for handling certain aspects of IRRASSPA. These utilities, when coupled with the capabilities of the four prior software tools, will serve to fill in the gaps and equalize the balance among the various components of IRRASSPA.

Following is a list of the various types of Database Utility software which might be obtained by a company.

- Backup/recovery
- Password security
- Incription/description software
- Image management software (text, graphics)
- Audit trail utilities
- Database tuning utilities
- Database development aids
- Database reloading and reorganizing aids
- Database sizing and responsiveness aids

All organizations need all of these facilities. However, it is important that these tools be part of any data management environment. Their interrelationships with the various components of IRRASSPA are self-explanatory.

#### *Data communications (DC) facilities*

It is not uncommon to hear data management referred to with the phrase DB/DC (Data Base/Data Communications). This terminology was introduced by IBM in the early 1970's to describe their thrust toward on-line database capabilities. In the early days of database, IBM was the only organization



to foresee that database effectiveness was a function of the on-line accessibility of the database to the user. Therefore when IBM introduced its concept of database, they did it using additional software facilities specifically intended to provide on-line accessibility to the database. It is IBM, therefore, which actually introduced the concept of data communications (DC) as an integral part of data management technology.

What originally started out as DC software intended to provide on-line accessibility to databases has now evolved to the point of providing the capability for *distributing* databases. Ultimately, these same software facilities provide the capability for establishing network DBMS facilities.

Data Communications facilities come in all shapes and sizes. They are used to provide accessibility to DDL's, DML's and DQL's as well as to some of the Database Utility tools. Data Communications software can be provided by either a DBMS vendor or a mainframe vendor, and it can be used for many things beyond just accessing databases. DC strategies are critical in determining the ultimate effectiveness of the data management environment. They affect all areas of IRRASSPA. As a result, data processing management must be careful in defining its DC architecture, making sure that the facilities adopted for Data Communications augment their strategic data management plans. When building the overall data management architecture, Data Communications facilities can be one of the most critical aspects in determining future capabilities, constraints, costs and productivity.

As I stated at the outset of this section, my topology of data management software tools is relatively arbitrary. I do not even claim that it is complete. However, I do think that it is useful in describing the types of facilities which one must consider in establishing a data management environment. To ignore any of these areas is to create downstream surprises and problems. I am also trying to get across the point that there is no single one best configuration for data management

software and that no DP department should depend totally on one vendor to provide all of its resources.

## CONCLUSION

I have tried to describe the world of data management in such a way that those who desire to enter it can see its overall implications. I admit to over-simplification, but hopefully, I have raised just as many questions as I have answered. That was my objective.

In this effort, I introduced several brain teasers. First, I introduced the concept of IRRASSPA. This was to focus the discussion and to define data management in terms meaningful to its implementation. Second, I defined three dimensions of data management, all of which affect IRRASSPA. In so doing, I redefined two well-known dimensions (data management structures and data management tools) so that they are now more meaningful. And finally, I introduced the crucial third dimension: Database Software Engineering, which, in my opinion, is of equal importance to the other two dimensions within the overall concept of data management. In fact, the Database Software Engineering methodology is of so much importance that it probably should be the *first step* in any effort to convert from a conventional data processing environment to a modern data management environment. Establishing an appropriate Database Software Engineering methodology will establish the foundation for customizing the other two dimensions of data management: data management tools and data management structure. I based this recommendation on my experiences and observations which say that if the inertia in an organization is so strong that the software engineering methodology cannot be changed, that organization will probably *never* get into the world of Data Management. Instead, what will happen, is that the organization will implement new structures and new tools, but it will end up using them in old ways.

## Data Base Design

In the past, data base design has been treated in literature and technology as a “universally” definable endeavor susceptible to some common or singular approach or model. It is now being recognized, especially by data base designers and users, that data base design is not nearly as tractable as it once appeared since its design is much more user- and environment-dependent than previously recognized.

The two sessions in this technical area present a broad range of views and approaches to designing data bases. They cover user experiences with specific data base management systems, and academic and industry research addressing data base design techniques and tools.

One session, Data Base Practicum, co-chaired by Prof. Jeffrey Hoffer and Donna Shepard Rund, surveys user experiences with various commercially available data base management systems and with custom data base design efforts.

The other session, co-chaired by Dr. David Jefferson and Nan Shu, will discuss data base design focusing on new techniques. These include new specification methodologies; representations of “properties” and their relationships; and other languages and tools for data base design. We find that there are key aspects we must examine when we are designing data base systems so that we can optimize the operational usage and realize the cost benefits for the following:

- the industry/company operational and organizational environment;
- the technical environment;
- the user’s current needs, wants, uses, and expectations of the system;
- the *future*, unplanned uses of the system.

The trend in data base design is toward a more formal, structured design approach. Success in this endeavor should result in more flexible and dynamically expandable data base systems.



Vincent Lum  
*Area Director*



# Properties of relationships and their representation

by RAMEZ EL-MASRI and GIO WIEDERHOLD

Stanford University  
Stanford, California

## INTRODUCTION

Data models can provide powerful abstractions to aid in the design of data structures that are relevant to database systems. A large amount of effort has been expended in the development of suitable data models. Several of these models distinguish between classes of entities and relationships among classes of entities in the abstractions they use for modelling the data. Among these models are the network model,<sup>1,2</sup> Schmid and Swenson's model,<sup>3</sup> the entity-relationship model,<sup>4</sup> Navathe and Schkolnick's model,<sup>5</sup> the semantic data model,<sup>5,7</sup> and the structural model.<sup>8,9,10</sup>

The relational model<sup>11</sup> does not explicitly include entity classes and relationships, but concentrates on a simple and uniform representation for all structures as relations. Relationships can then be discovered at query processing time by using relational operators such as the JOIN operator.

The hierarchical model<sup>12</sup> represents only 1:N relationships in a straightforward manner, and requires additional reference structures and redundant record types to represent M:N relationships.

We are motivated by two reasons to consider that the correct representation of properties of relationships is important. First, the data model is a representation of some real-world situation, and should reflect as many properties that are known about the real-world situation as possible. Second, most database implementations can take advantage of known relationships among entity classes, and hence it is useful to represent such relationships in the data model. Database processes also take advantage of relationships. Examples include all of the hierarchical and network data languages, but relational languages may also recognize relationships in the database structure. For instance, the relational language SEQUEL<sup>2,13</sup> as used in System-R has a LINK statement to relate tuples from different relations. The language LSL<sup>14</sup> is a general query language for relational, hierarchical, and network data models which utilizes relationships.

All the models cited above, with the exception of the relational model, offer rules that govern existence dependencies of represented entities from different entity classes that are related together by a relationship. The pure relational model does not define such rules, but extensions of the model provide for the definition of arbitrary assertions

to state such rules in an integrity subsystem separate from the model itself.<sup>15,16</sup> Such integrity constraints are not restricted to relationships, and can define arbitrary rules to govern the behavior of a data model. However, it is useful to explicitly represent the important relationship-oriented constraints that have a natural correspondence to real-world structures, and that can be helpful in designing an implementation of the data model. We hence do not consider intertuple constraints as: if sex = male then pregnancies = 0, nor constraints with procedural semantics: managers earn more than their employees. Such constraints will still require separate integrity assertions. We are concerned with constraints of the form: an inventory item must have a supplier.

## ENTITY CLASSES AND RELATIONSHIPS

The concept of an entity class is often used in database models.<sup>2,4,5,6,10</sup> An *entity class* is a set of objects of similar structure. For example, an entity class *cars-in-California* is the set of all cars registered in California, or an entity class *car-manufacturers* is the set of all car manufacturers.

Each object is described by properties (or attributes) that the object shares with all other objects of the entity class. For example, the entity class *cars-in-California* may have the properties *license-number*, *color*, *owner*, *make*, *year*, while the entity class *car-manufacturers* may have the properties *manufacturer-name*, *location*. Some properties have unique values for each object in the class, and hence may serve to identify each object in the class uniquely. In our example, *license-number* and *manufacturer-name* identify uniquely a *car* and *manufacturer* respectively.

Clearly, some object may not have all the properties, or may have additional properties that describe them, but all our formal models will impose such a regularity. Not much damage is done: an inappropriate attribute can be given a value *not applicable*, and exceptional attributes are rarely useful for computational purposes. Subclass concepts can deal with subsets of entities that have additional properties,<sup>5,6,10</sup> but this is not directly relevant to our discussion.

A *relationship* between two entity classes is a mapping that associates with each object of one entity class a number of objects (possibly none) of the other entity class. For example, a relationship *car:manufacturer* associates with each

*car* object a related *manufacturer* object such that the car is made by this manufacturer.

A relationship has rules that govern the mapping of objects from the two entity classes. We shall call these rules the properties of the relationship (different from the use of the word *property* for entity classes). For example, the *car:manufacturer* relationship may have the following properties:

1. Every car must be related to exactly one manufacturer.
2. A manufacturer is related to at least one car, but can be related to any number of cars.

These two rules imply that the relationship is a mutual or total dependency: the existence of a car depends upon the existence of its manufacturer, and the existence of a manufacturer depends upon his having manufactured at least one car. The rules also imply that a manufacturer can be related to  $N$  cars, when  $N$  is unspecified. Hence, the cardinality of the relationship *cars:manufacturers* is  $1:N$ .

#### PROPERTIES OF RELATIONSHIPS

There are two important properties of a relationship between two entity classes: the *cardinality* and the *dependency*. We first consider the cardinality property. Within each cardinality case, we will discuss the dependency property.

The cardinality property of a relationship places restrictions on the number of objects of one entity class that may be related to an object of the other entity class. The dependency property governs whether an entity can exist independently, or whether it requires the existence of related entities from another entity class. We will see the importance of the cardinality and dependency properties of a relationship when we discuss the representation of relationships. In the ensuing discussion, we will use  $A$  and  $B$  to denote two classes of entities, and  $A:B$  to denote a relationship between the two entity classes.

Cardinalities are classified into three types:  $1:1$ ,  $1:N$  and  $M:N$ . Dependencies are classified into four types: total, partial  $A$  on  $B$ , partial  $B$  on  $A$ , and no-dependency.

We now discuss the dependency properties as they apply to a relationship of known cardinality.

##### *One-to-one relationships*

Consider a  $1:1$  relationship  $A:B$ . We can distinguish four cases based upon the dependency properties of the objects in the two related entity classes.

- (a) Total dependency: Every object of class  $A$  must be related to one object of class  $B$ , and vice versa. In this case, a one-to-one correspondence exists between the objects in the two entity classes.
- (b) Partial dependency of  $A$  on  $B$ : Every object of entity class  $A$  must be related to an object of class  $B$ , but some objects of class  $B$  can exist that are not related to an object of class  $A$ . Mathematically, this relation-

ship is defined by a total  $1:1$  function from  $A$  into  $B$ . The inverse is a partial  $1:1$  function from  $A$  onto  $B$ .

- (c) Partial dependency of  $B$  on  $A$ : This is symmetric to case (b) above.
- (d) No-dependency: Objects can exist in both entity classes that are unrelated to an object of the other class. This relationship is defined by two partial  $1:1$  into functions that are inverses of one another.

These properties are relevant in the definition of the semantics of one-to-one relationships. We give examples to illustrate this point.

A *marriage* relationship between the entity classes *husbands* and *wives* is a  $1:1$  total dependency. A relationship *managers:departments* is a partial dependency of *managers* on *departments* (every *manager* must be related to a *department* but short periods of time may exist when a *department* does not have a *manager*). Finally, a relationship *merchant ships:captains*, which describes the current assignment of a *captain* to a *ship* is a no-dependency.

##### *One-to-N relationships*

Consider a relationship  $A:B$  of cardinality  $1:N$ . Here, an object of class  $B$  can be related to at most one object of class  $A$ , while an object of class  $A$  can be related to any number of class  $B$  objects. If  $N$  is specified as a number, this restricts the number of class  $B$  objects related to a class  $A$  object to a maximum of  $N$ . We can distinguish four types of existence dependencies for such a relationship.

- (a) Total dependency ( $i$ ): Every object of class  $A$  must be related to at least  $i$  objects,  $i > 0$ , of class  $B$ , and every object of class  $B$  must be related to exactly one object of class  $A$ . The relationship is defined by a total function from  $B$  onto  $A$ .
- (b) Partial dependency ( $i$ ) of  $A$  on  $B$ : Every object of class  $A$  must be related to at least  $i$  objects,  $i > 0$ , of class  $B$ . An object of class  $B$  may be related to at most one object of class  $A$ . The relationship is defined by a partial function from  $B$  onto  $A$ .

Note that we define the dependency of class  $A$  objects on class  $B$  objects for a  $1:N$  relationship  $A:B$  by requiring an object of class  $A$  to be related to at least  $i$  objects of class  $B$ ,  $i > 0$ . We do not define it by requiring an object of class  $A$  to be related to exactly  $i$  objects, since the latter case can be decomposed into  $i$  relationships of cardinality  $1:1$ .

- (c) Partial dependency of  $B$  on  $A$ : Every object of class  $B$  must be related to exactly one object of class  $A$ . The relationship is defined by a total function from  $B$  into  $A$ .
- (d) No-dependency: Objects can exist in either class that are unrelated to objects of the other class. The relationship is defined by a partial function from  $B$  into  $A$ .

For examples, see the complete report.<sup>17</sup>

*M-to-N relationships*

Finally consider a relationship  $A:B$  of cardinality  $M:N$ . This is the general case. No restrictions exist on the number of objects related to an object of either class. If  $M$  or  $N$  or both are specified by numbers, a restriction on the maximum number of objects related to a single object of the other class is specified as in the  $1:N$  case. We can again distinguish four types of existence dependencies for such a relationship.

- (a) Total dependency ( $i,j$ ): An object of class  $A$  must be related to at least  $i$  objects of class  $B$ ,  $i>0$  while an object of class  $B$  must be related to at least  $j$  object of class  $A$ ,  $j>0$ .
- (b) Partial dependency ( $i$ ) of  $A$  on  $B$ : An object of class  $A$  must be related to at least  $i$  objects of class  $B$ ,  $i>0$ .
- (c) Partial dependency ( $j$ ) of  $B$  on  $A$ : An object of class  $B$  must be related to at least  $j$  objects of class  $A$ ,  $j>0$ . This case is symmetric to (b).
- (d) No-dependency: No restrictions exist on the relationship. This is the most general case.

An example of a partial  $M:N$  dependency is the relationship between the two entity classes *bills-passed-in-congress* and *congress-persons* that relates each passed bill with the congress-persons who voted yes on the bill. If we assume that a particular congress has 100 members, and a passed bill requires at least 51 yes votes, the relationship *bills-passed-in-congress:congress-persons-voting-yes* is a partial dependency ( $i$ ) of *bills-passed* on *congress-persons* of cardinality  $M:N$ , with  $i=51$ .

An example of a no-dependency  $M:N$  relationship is that between the entity class *suppliers* that can supply parts to a company, and the entity class *parts* that are used by the company. Suppliers can exist that do not supply any parts at some moment, and parts may exist for which there is no longer a supplier. The relationship relates each *supplier* object with the *part* objects he currently supplies.

REPRESENTATION OF RELATIONSHIPS OF KNOWN PROPERTIES

A data model is used to represent a real-world situation. The objects in the real-world continuously undergo change. There are many constraints on the way in which the real-world can change. Since a data model implies the specification of rules, it is best if these rules match real-world constraints. We expect representations of objects from entity classes to be inserted into and deleted from the database. These insertions and deletions should be governed by the rules implied in the data model.

A relationship relates objects from entity classes. The properties of relationships discussed in the previous section define rules that govern changes to related objects from two entity classes that have a relationship between them.

These are the rules we wish to consider in this section. When properties of a relationship are known beforehand, it is useful to represent these properties in the model as rules

so that updates to the objects that participate in a relationship will maintain these properties. If properties of a relationship are not known, it can be represented in the data model using the most general case: the no-dependency,  $M:N$  relationship.

While integrity assertion statements applied to a model are sufficient to describe any rules that constrain the model, they cannot affect the structure of the model, since they are not a part of the data model. As we shall see, known properties of relationships suggest specific model structures, and a small set of rules expressed in the data model can represent all the properties of relationships discussed in the previous section.

*Relationships and their constraints in normalized models*

An entity class is represented in relation-based models as a set of *tuples* of identical structure that constitute a *relation*. Each tuple represents an object from the entity class. The properties of the entity class are described by the *attributes* of the relation. The attributes define the domains from which data items in a tuple can take values. The values of the identifying, or *key*, attributes in a tuple define the correspondence between a real-world object and that tuple. Hence, values for key attributes are required to be unique for each tuple in the relation. The non-identifying, or *dependent* attributes, represent other properties of the object.

In some cases, more than one set of identifying attributes exist. We will assume that one set is designated to identify the objects of the entity class, the primary key (*PK*), and use that set in our discussion of the representation of relationships.

Relationships can be represented in several ways. The three-relation representation uses a separate relation to represent each entity class. A third association (or relationship) relation, which contains the *PK* attributes of the other two relations, represents the relationship.<sup>3,4,8</sup> A tuple in this association relation serves to associate tuples from the relations that represent the entity classes.

Figure 1(a) shows an example. Two relations that repre-

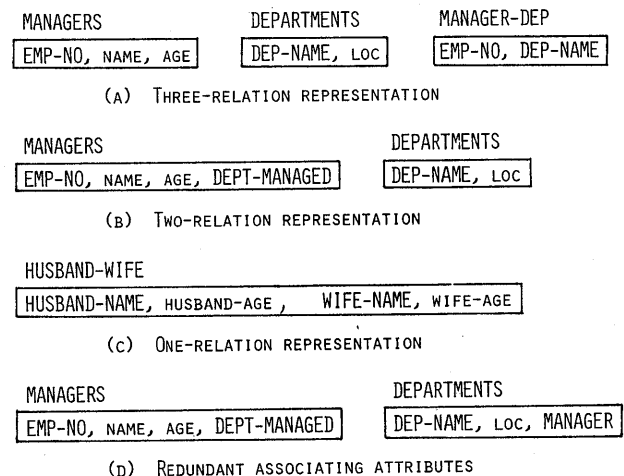


Figure 1—Alternative representations of relationships.

sent entity classes *managers* and *departments* are related via a third relation *manager-dep*. We call *manager-dep* the associating relation, and the attributes *emp-no* and *dep-name* in *manager-dep* the associating attributes. A tuple (55, Payroll) in *manager-dep* means employee number 55 is the manager of the Payroll department, and associates the tuple with PK 55 in *managers* with the tuple with PK Payroll in *departments*. In our diagrams, we will show the PK and the associating attributes in capital letters.

The two-relation representation includes the associating attributes in one of the relations that represent an entity class. Figure 1(b) shows an example, using the same *manager:department* relationship. This representation only works for relationships of cardinality 1:1 or 1:N, since only one value of the associating attribute (the *dept-managed* attribute in our example) can exist in a normalized relation.

For 1:1 relationships, a one-relation representation is possible (Figure 1(c)). It is also possible to represent the associating attributes twice, once via the associating attribute *dept-managed* in *managers* and a second time via the associating attribute *managers* in *departments*. One of the associating attributes is redundant, and to assure that a *manager* tuple is associated with the same *department* tuple in both representations, an additional constraint is needed.

Since a large number of integrity constraints can be costly to manage, one measure of *goodness* for the representation of a relationship is that it requires a minimum of additional constraints, but still correctly represents the properties of the relationship.

When tuples that participate in a relationship are inserted or deleted, the transformation must not lead the database from a consistent state into an inconsistent one. The properties of the relationship define conditions for a consistent state. Hence, update constraints that maintain the properties of a relationship should be specified in the data model. We can consider updates of the database as transactions which transform one consistent state of the database into another.

Update constraints are specified on the relations of the data model to reflect the properties of the relationship. We now discuss these constraints for representations of relationships between two entity classes.

#### The associating attributes (AA) constraint, and rules to maintain it

The associating attributes constraint ensures that we are associating two existing tuples (that represent two existing entities) in the database: it does not make sense to associate a *department* which is not in the database with a *manager*. Hence, this AA constraint specifies that the values of associating attributes must correspond to values of primary keys in the relations that represent the two related entity classes.

The consequences of this constraint for insertion and deletion of tuples are:

- (1) Insertion of a tuple in a relation with associating at-

tributes is permitted only if the values of the associating attributes correspond to values of primary keys of tuples in the relations that represent the entity classes.

- (2) Deletion of a tuple from a relation that represents an entity class is permitted only if the value of the primary key does not correspond to a value of associating attributes in some tuples that remain in the database after the deletion.

The insertion rule is straightforward. Attempts to insert a tuple which violates the rule are rejected. To remedy the situation, one must first insert other tuples that make the required insertion legal. Another possibility is to insert this tuple, and other tuples associated with it simultaneously as a transaction so that after all tuples in the transaction are inserted, the insertion rule holds. The whole transaction may be rejected if it results in a violation.

For deletion, two rules can be specified to ensure the consistency. The *prohibit deletion* (or *PD*) rule aborts the deletion until the tuples with associating values that correspond to the PK of the tuple being deleted are explicitly deleted. The *delete related tuple* (or *DLT*) rule carries out the deletion, and automatically deletes all tuples with associating values that correspond to the PK of the tuple being deleted to ensure the consistency. This may result in the deletion of tuples that represent objects from the other entity class.

#### The cardinality constraint

We now consider how cardinality constraints can be expressed in a data model. Consider the three-relation representation. If we specify no cardinality constraints, it represents a general *M:N* relationship (Figure 2(a)). For a 1:N relationship if we restrict the associating attribute *emp-no* in the relation *dept-emp* to have unique values in the different tuples of *dept-emp* at all times, the 1:N cardinality is guaranteed. To further restrict the cardinality to 1:1, we specify that both associating attributes have unique values also (Figure 2(c)). In our diagrams, we specify this uniqueness constraint by a (*U*).

The two-relation representation can represent a 1:N or a 1:1 cardinality. Figure 3(a) shows a 1:N relationship. If we restrict the associating attribute to unique values (Figure 3(b)) we get a 1:1 cardinality for the relationship. Note that the UA constraint can also be used to specify the PK when it is a single identifying attribute of a relation that represent entity classes (Figure 2 and 3).

#### The dependency constraints

Dependency constraints of a relationship *A:B* specify whether an object from one of the entity classes can exist independently, or whether the object must be related to objects from the other class at all times.

A no-dependency relationship is best represented by the three-relation representation. Tuples then exist independ-

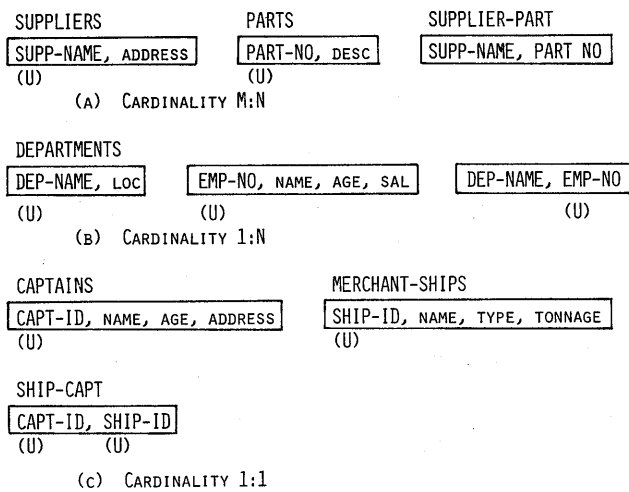


Figure 2—Cardinality constraints for the 3-relation representation.

ently in the relations that represent the entity classes, and are associated via the third relation. This works for 1:1, 1:N, and M:N cardinalities (Figure 2).

A partial dependency for a relationship of cardinality 1:1 is directly specified by the two-relation representation (Figure 3(b)). No additional constraints are needed, since the AA constraint enforces the partial dependency. The two-relation representation can also represent a partial dependency of B on A for a 1:N relationship A:B. For example, Figure 3(a) shows a partial dependency of employees on departments. The partial dependency of A on B cannot be represented without additional constraints, which we now consider.

As an example, consider the 1:N relationship departments:employees and suppose the relationship is a partial dependency (i) of departments on employees. Hence, a department is always related to at least i employees, where i is a specified positive number, but employees may exist independently. We represent an additional constraint on the three-relation representation (Figure 4) by a [min i] on the relating attribute dept-name in dept-emp. This constraint means that for every department tuple in departments, at least i tuples exist in dept-emp with the same value for dept-name.

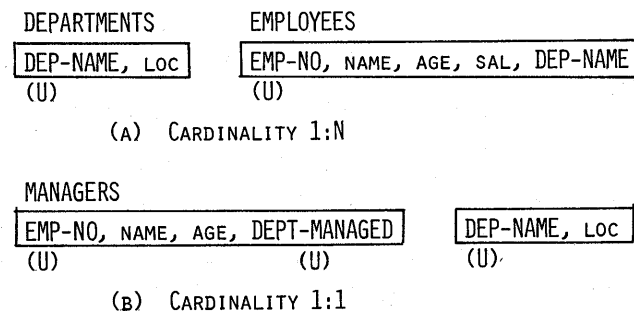


Figure 3—Cardinality constraints for the 2-relation representation.

To maintain such a constraint, insertion of a new department tuple must specify at least i employee tuples that already exist in employees, and a transaction insert-new-department will insert the new department and the least i associating tuples in dept-emp. Insertion of an employee is unconstrained, since it will never cause an inconsistency.

Deletion of a department or employee tuple can be governed by either the PD or the DLT rules. Deletion of an associating dept-emp tuple can result in an inconsistency if it reduces the number of employees tuples associated with a department tuple to less than i. We can either specify the PD rule (prohibit deletion of the associating tuple and do not carry out the requested deletion transaction) or the DLT rule (delete the department tuple, and all tuples in dept-emp associated with it). In this case, the PD rule is more plausible, and can affect that the enterprise reconsider the state of its departmental structure.

A partial dependency of a relationship of cardinality M:N can be handled in an identical way to the 1:N partial dependency. The only difference is the absence of the (U) specifying the cardinality on the associating attribute emp-no in dept-emp. Total dependencies are handled similarly, but require two rules for maintaining the consistency.

Non-normalized relations

If we allow relations that are not in first normal form<sup>11</sup> a relation may include a repeating attribute. A repeating attribute allows a set of values for the attribute in a tuple of the relation rather than a single value. Now 1:N and M:N relationships with partial or total dependency can be represented more naturally using the two-relation representation. This is because we can allow the associating attribute to associate a single tuple to a set of tuples of the other relation.

Consider the partial dependency (i) of departments on employees in a 1:N relationship departments:employees. It can be represented as in Figure 5(a). The associating attribute emp-no in departments is a repeating attribute, and the (i) means at least i values for this attribute must exist in each tuple of the departments relation. The (U) specifies here that values of the attribute in all the relation are unique, and hence ensures the 1:N cardinality.

A partial dependency relationship of cardinality M:N can be represented similarly (Figure 5(b)) with the (U) removed to allow the M:N cardinality.

COMPARISON OF THE REPRESENTATION OF RELATIONSHIPS IN MODELLING APPROACHES

In this section, we compare the representation of relationships in the network model, the entity-relationship model, Schmid and Swenson's model, Navathe and Schkol-

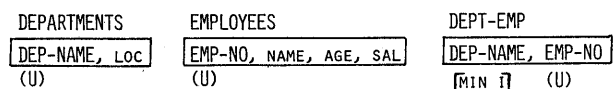


Figure 4—A partial dependency (I) of departments on employees.



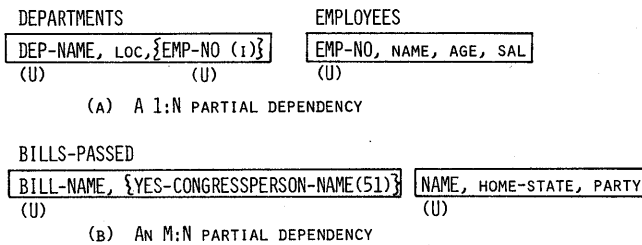


Figure 5—Representation using non-normalized relations.

nick's model, the semantic data model, and the structural model. We show what choices each model offers for the representation of relationships, and for maintenance of the consistency of the representation.

We do not include the basic relational model since it does not explicitly represent relationships in the model, but requires integrity assertions separate from the model to represent relationship constraints. While this is completely general, it does not influence the design of the data model, and may be expensive to maintain in an implementation.

#### The network model

The network model uses record types to represent entity classes, and link-sets to represent 1:N relationships. To represent an M:N relationship between two record types, two link-sets and an additional link record type are defined.<sup>2</sup>

The CODASYL syntax allows specifications of database procedures that are automatically invoked upon update and deletion. These procedures could be used to implement arbitrary integrity assertions, but cannot be considered part of the model, and not all current implementations provide for them.

There are several types of link-sets. An automatic link-set from owner record type *A* to member record type *B* represents a 1:N relationship *A:B* which is a partial dependency of *B* on *A*, since every record in *B* must be linked to a record in *A*. No associating attributes are represented, but the AA constraint is implicit. It is not clearly specified how it is maintained, but, in all network systems we know of, the DLT rule is applied, since loss of linkage makes the record inaccessible. However, this is not explicitly mentioned in the CODASYL report.<sup>1</sup> To achieve explicit control over this process, the maintenance of the linkage has to be programmed appropriately.

A manual link-set from *A* to *B* represents a no-dependency relationship *A:B* of cardinality 1:N, since member records in *B* may exist that are not linked to an owner record. No associating relation is used (as in Figure 2(b)), and it is not explicitly mentioned in<sup>1</sup> what action is taken when an owner record is deleted. The choice of whether the member records are deleted or become records without an owner has to be specified outside of the model.

An M:N relationship *A:B* represented by two automatic link-sets and a link record is a no-dependency M:N relationship, with the link record type representing the relationship.

Direct representation of 1:1 relationships in the network

model is not possible within the model, and has to be defined by database procedures.

#### The entity-relationship model

The entity-relationship model<sup>4</sup> uses entity relations to represent entity classes, and relationship relations to represent relationships. The entity-relationship diagram can specify the cardinality of a relationship explicitly by stating whether it is 1:1, 1:N, or M:N. Two types of entity relations are defined: regular entity relations and weak entity relations. A no-dependency relationship of any cardinality is directly specified by a relationship relation between any two regular entity relations. A partial dependency of *B* on *A* in a 1:N relationship *A:B* is specified by a weak relationship, where entity relation *A* is a regular entity relation and entity relation *B* is a weak entity relation. The DLT rule is used to maintain the consistency in both cases.

#### Schmid and Swenson's model

Schmid and Swenson's model<sup>3</sup> uses third normal form relations. Entity classes are represented by independent object types, and relationships are represented by association relations. There are three other types of relations in the model, but we only consider independent object type and association relations, since they are used to represent relationships between entity classes. An association represents a general M:N no-dependency relationship. The AA constraint is maintained by the PD rule. Hence, associating tuples must be explicitly deleted before deleting the independent object type tuple. Representation of 1:1 and 1:N relationships require additional constraints, separate from the model, as do dependency constraints.

#### Navathe and Schkolnick's model

Navathe and Schkolnick's model uses entities to represent entity classes and simple associations to represent relationships. Other types of associations exist. Since we are here only considering relationships between different entity classes, and not subclass relationships, we will only consider simple associations. Two types of connectors exist in this model: directed and undirected.

We will categorize the three examples of simple associations given in.<sup>5</sup> A simple association between two entities *A* and *B*, where the connectors are not directed, defines a no-dependency M:N relationship *A:B*. Consistency is maintained by the DLT rule (deletion of association tuples when an entity is deleted).

An owner-member simple association, with directed connectors from the owner entity *A* to the association, and from the association to the member entity *B*, defines a 1:N relationship *A:B*. The dependency is a partial dependency of *B* on *A*, since tuples cannot exist in *B* that are not related to tuples in *A*. The consistency is maintained by the DLT rule.

A simple association with an undirected connector from entity *A* to the association, and a directed connector from the association to entity *B*, specifies an M:N relationship

$A:B$  that is a partial dependency (1) of  $B$  on  $A$ , since each entity in class  $B$  must be related to at least one entity of class  $A$ . To maintain the consistency, when an entity of class  $A$  is deleted that is the only entity related to a particular entity of class  $B$ , this related class  $B$  entity is also deleted, which is the DLT rule.

Additional integrity assertions, separate from the model, may be needed to specify other cardinalities and dependencies.

*The semantic data model*

The semantic data model<sup>6,7</sup> is a rich semantic model which supports powerful user interface facilities. It provides concrete object classes to support entity classes, and inter-class connections and event class to represent relationships. Many semantic constructs are part of the model, including subclasses, abstractions, aggregates, and events. Detailed specification is provided for attributes, such as mandatory (null values not allowed), unique, and ordering of values. Multi-valued attributes are permitted.

We will discuss here what types of relationships inter-class connections can support. Suppose a relationship  $A:B$  is represented by an attribute in the concrete object class  $A$  that references objects from concrete object class  $B$ . If the attribute is specified to be single-valued and unique, the relationship is 1:1, and if it is specified to be single valued and non-unique, the relationship is  $N:1$ . If the attribute is a set (repeating attribute) and unique, the relationship is 1: $N$ , and if it is a set but not unique, the relationship is  $M:N$ .

For single-valued attributes that are mandatory (no null or unknown value), the relationship is a partial dependency of  $A$  on  $B$  for 1:1 or  $N:1$ . For a mandatory set attribute, a partial dependency (1) is specified for 1: $N$  or  $M:N$ . Total dependencies and partial dependencies ( $i$ ) require additional constraints, separate from the model.

The consistency of the AA constraint is maintained using the PD rule, since objects in class  $B$  that are referenced by objects from class  $A$  may not be deleted.<sup>7</sup>

*The structural model*

The structural model<sup>10</sup> is formed from relations in Boyce-Codd normal form and three types of connections: ownership, reference, and identity connections. Identity connections are used to represent subrelations of existing relations, and carry out functions similar to the categorization, selection, and subsetting associations of Navathe and Schkolnick's model, and the subclasses of the semantic data model, so we will not consider them further. Sets of attributes can be defined to have unique values, using the (U) notation described above.

A partial dependency of  $B$  on  $A$  of cardinality 1: $N$  is specified either using a reference connection (Figure 6(a)) or an ownership connection (Figure 6(b)). We show the 1: $N$  relationship *departments:employees* with partial dependency of *employees* on *departments*. The reference connection maintains the AA consistency using the PD rule, while the

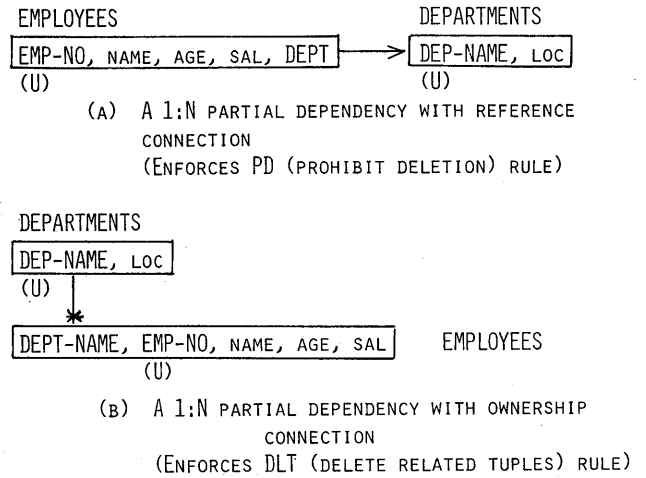


Figure 6—Partial dependencies in the structural model.

ownership connection maintains it using the DLT rule. Both connections associate a *department* tuple with  $N$  *employee* tuples, and hence represent a 1: $N$  relationship. The 1:1 relationship can be specified by restricting the associating attribute *dept* in the *employees* relation to unique values.

No-dependency relationships are represented using three relations and two connections. A 1:1 or 1: $N$  cardinality is specified by restricting the associating attributes to unique values, otherwise the relationship is  $M:N$ . Figure 7 shows the 1: $N$  no-dependency relationship *department:employees*. An ownership or reference can be used to specify DLT or PD for each entity class.

By choice of ownership or reference connections, the consistency maintenance rules are specified. In Figure 7(a),

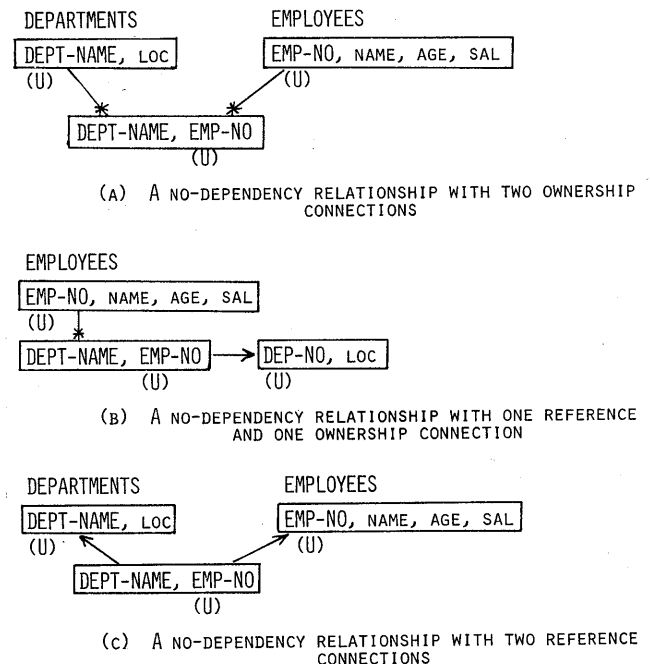


Figure 7—No-dependencies in the structural model.

deletion is unrestricted for both employee and department tuples, and the AA consistency maintained using the DLT rule. In Figure 7(b), deletion is unrestricted for employees (DLT), and restricted for departments associated with at least one employee (PD). In Figure 7(c), the PD rule is used for both department and employee tuples. The choice of representation depends upon whether a check is desired when tuples that are related to other entity classes are deleted.

Total dependencies, and partial dependencies (*i*) for 1:N and M:N relationships, cannot be represented in the structural model.

SUMMARY AND POSSIBLE EXTENSIONS

None of the above models allows a full choice for representation of the properties of relationships discussed. In particular, partial dependency (*i*),  $i > 1$ , for 1:N and M:N relationships, and total dependency relationships, cannot be represented without the specification of constraints separate from the model.

Table I shows what relationship properties each of the six modelling approaches can represent without additional integrity constraints.

An extension to the structural model can allow representation of all the relationship properties discussed earlier. We can suggest similar extensions for other models, and perhaps some of them have already been considered by their authors. The structural model is the one we are most familiar with.

We attach a constant number  $\langle i \rangle$  to an ownership or a reference connection. This means *at least i tuples from the relation at the N side of the connection are attached to each tuple at the 1 side of the connection at all times*. If two numbers  $\langle i1 - i2 \rangle$  are attached to the connection, a minimum of *i1* and maximum of *i2* tuples are allowed. The default is  $\langle 0, \text{infinity} \rangle$ . Now, all cardinalities and dependencies discussed can be represented.

CONCLUSIONS

In this paper, we examined the semantic properties of a relationship between two entity classes. We identified two

TABLE I.—Relationship properties for different modelling approaches

		NETWORK MOD	ENT- RELAT	SCHMID AND SW	NAV SCHKOL	AND DM	SEM DM	STRUCT DM
1:1	NO-DEP		X				X	X
	PARTIAL						X	X
	TOTAL							
1:N	NO-DEP	X	X		X	X	X	X
	PARTIAL	X	X		X	X	X	X
	PART(I)				X(*)	X(*)		
	TOTAL							
M:N	NO-DEP	X	X	X	X	X	X	X
	PART(I)				X(*)	X(*)		
	TOT(I,J)							
	DLT		X		X			X
	PD			X		X		X
(*)	FOR I=1 ONLY							

properties: the cardinality and the dependency, and discussed the different cases of each property.

We then showed how these properties can be represented using normalized relations, and what update constraints are necessary to maintain these properties. We identified the basic AA (associating attribute) constraint and described two rules to maintain it: DLT (delete related tuples) and PD (prohibit deletion). We then discussed the cardinality and dependency constraints.

We compared six data models with respect to the choices they offer for the representation of relationships. We subsequently showed how extensions can represent all properties of relationships discussed. We suggest that the following dimensions be used in describing relationships in data models:

- (1) Cardinality: 1:1, 1:N, or M:N
- (2) Dependency: Total, partial (2), no-dependency.
- (3) Deletion rule to maintain consistency: DLT or PD

ACKNOWLEDGMENT

This work was supported by the Defense Advanced Research Projects Agency, Contract MDA 903-77-C-0322, under the KBMS Project.

REFERENCES

1. CODASYL Data Description Language, *Journal of Development* (June 1973); NBS Handbook 113, January 1974, 155pp.
2. Taylor, R. W. and Frank, R. L., "CODASYL Data-Base Management Systems," *ACM Comp. Surv.* 8(1), March 1976, pp. 67-104.
3. Schmid, H. A. and Swenson, J. R., "On the semantics of the relational model," *ACM SIGMOD 1975 Conf.*, pp. 211-223.
4. Chen, P. P. S., "The Entity-Relationship Model—Towards a Unified View of Data," *TODS* 1(1), March 1976, pp. 9-36.
5. Navathe, S. B. and Schkolnick, M., "View Representation in Logical Database Design," *ACM SIGMOD 1978 Conf.*, pp. 144-156.
6. Hammer, M. and McLeod, D., "The Semantic Data Model: A Modelling Mechanism for Data Base Applications," *ACM SIGMOD 1978 Conf.*, pp. 26-36.
7. McLeod, D., "The Semantic Data Model and Its Associated Structure User Interface," Ph.D. Thesis, MIT/LCS/TR-214, August 1978, 378 pp.
8. Wiederhold, G., *Database Design*, McGraw-Hill, 1977.
9. El-Masri, R. and Wiederhold, G., "Data Model Integration Using the Structural Model," *ACM SIGMOD 1979 Conf.*, pp. 191-202.
10. Wiederhold, G. and El-Masri, R., "A Structural Model for Database Systems," Report CS-79-722, Stanford University, February 1979, 53 pp.
11. Codd, E. F., "A Relational Model for Large Shared Data Banks," *CACM* 13(6), June 1970, pp. 377-387.
12. Tsichritzis, D. C. and Lochovsky, F. H., "Hierarchical Data-Base Management," *ACM Comp Surv.* 8(1), March 1976, pp. 105-124.
13. Chamberlin, D. D. et al., "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control," *IBM Jour. R&D* (20) 6, November 1976.
14. Tsichritzis, D., "LSL: A Link and Selector Language," *ACM SIGMOD 1976 Conf.*, pp. 123-134.
15. Stonebraker, M., "High level integrity assurance in relational data base management systems," ERL-M473, U. C. Berkeley, May 1974.
16. Eswaran, K. P. and Chamberlin, D. D., "Functional Specifications of a Subsystem for Database Integrity," *VLDB 75 Conf.*, ACM.
17. El-Masri, R. and Wiederhold, G., "Properties of Relationships and Their Representation," CS Report, Stanford (to appear).

## Earth Resources

The purpose of this session is to present a cross-section of talks relating to the current state of computer simulation modeling as applied to solar energy systems. This field is of major importance today because of the national need to explore alternative energy sources in a quick, comprehensive, and inexpensive manner.

Since a large scale solar energy industry does not presently exist, the application of these models can save years of time that might otherwise be wasted on unfeasible systems. The models to be discussed deal with:

- Engineering and economic performance of solar thermal and photovoltaic power plants.
- The effects of ownership options, government policies and operating alternatives on the economic viability of small photovoltaic systems.
- The interaction of solar electric power plants with existing utility grids.



Leigh S. Rosenberg  
*Area Director*



# BALDR-1: A solar thermal system simulation

by JOSEPH G. FINEGOLD and F. ANN HERLEVICH

*Solar Energy Research Institute*  
Golden, Colorado

## INTRODUCTION

A system simulation, BALDR-1, was written to model the performance of solar thermal systems. The original application was to model the performance and economics of 0.1-10 MW<sub>e</sub> solar thermal electric power plants.<sup>1</sup> It has subsequently been used in receiver selective surface value analysis and in thermal storage value analysis, and is being adapted currently to model industrial process heat systems.

The FIELD code models the optical and thermal performance of the collector field and thermal transport subsystems. The POWER code models the power conversion and energy storage subsystems. The ECON code determines the initial capital cost of the power plant and the life-cycle busbar energy cost. A flow chart of the system simulation is shown in Figure 1.

## FIELD CODE

The FIELD code is a second-order simulation based on a similar code previously developed by the Aerospace Corporation with modifications by the Jet Propulsion Laboratory (JPL),<sup>2</sup> and by Battelle Pacific Northwest Laboratories (PNL).<sup>3</sup> The FIELD code uses meteorological data read in from SOLMET or TMY format weather tapes in 15-minute or hourly increments. Data used in the current version of FIELD are: direct normal insolation, solar time, global insolation, ambient temperature, dew point, and day of the year. The FIELD code models the performance of collector subsystems in four different ways depending on the type of collector subsystem being modelled. There are separate modules to calculate the optical and thermal performance of each generic collector type. If the need should arise to model other collector types, it is a simple matter to add additional optical and thermal performance modules.

For point focus central receiver systems (PFCR) and line focus central receiver systems (LFCR), the optical efficiency of the concentrator field is determined at each time step by a bivariate linear interpolation of tables of optical efficiency as a function of solar azimuth and zenith angles. These efficiency tables must be input and generally result from third-order simulation programs such as DELSOL<sup>4</sup> and MIRVAL.<sup>5</sup>

The radiative losses from the receiver are calculated in the

FIELD code based on the effective receiver temperature, the effective absorptivity and emissivity of the receiver and the effective normalized receiver area. The convective and conductive losses are assumed to be a constant fraction of the calculated radiative losses. The value of this fraction can be adjusted to yield receiver efficiencies similar to those predicted by third-order simulations and reconciled with experimental results. The energy incident on the receiver at each time step per unit area of collector is then equal to the product of the optical efficiency, direct normal insolation, and the time step. The energy collected at the receiver is this term minus the calculated thermal losses. The energy collected in the collector field (ECF) is then equal to the product of the energy collected at the receiver and the thermal transport efficiency.

For the point focus distributed receiver systems (PFDR), e.g., paraboloidal dishes, and fixed mirror distributed focus systems (FMDF), e.g., hemispherical bowls, the optical efficiency is determined by explicit calculation at each time step. This calculation includes the effects due to solar azimuth, zenith, concentrator position, intercept factor, reflectivity, blockage, shadowing, edge losses and dust. The receiver thermal losses are calculated in a manner identical to that described above for the central receiver systems. The energy incident on the receiver at each time step per unit area is again equal to the product of optical efficiency, direct normal insolation and the time step. The energy collected at the receiver is the energy incident on the receiver minus the thermal losses. The energy collected from the field (ECF) is the product of the energy collected at the receiver and the thermal transport efficiency. This may be determined per unit area of concentrator or per unit collector module.

For the line focus distributed receiver systems (troughs) with either tracking collectors (LFDR-TC) or tracking receivers (LFDR-TR), the optical efficiency is determined by explicit calculation at each time step. This calculation includes the effects due to solar azimuth, intercept factor, reflectivity, blockage, shadowing, edge losses, dust, secondary concentrator efficiency and transmissivity of receiver cover. The thermal losses of the receiver are based on a selectable fraction of the thermal losses resulting from tests of the best receiver to date.<sup>6</sup> This fraction allows for future improvements in receiver design such as selective coatings, evacuated covers, etc. The energy incident on the receiver

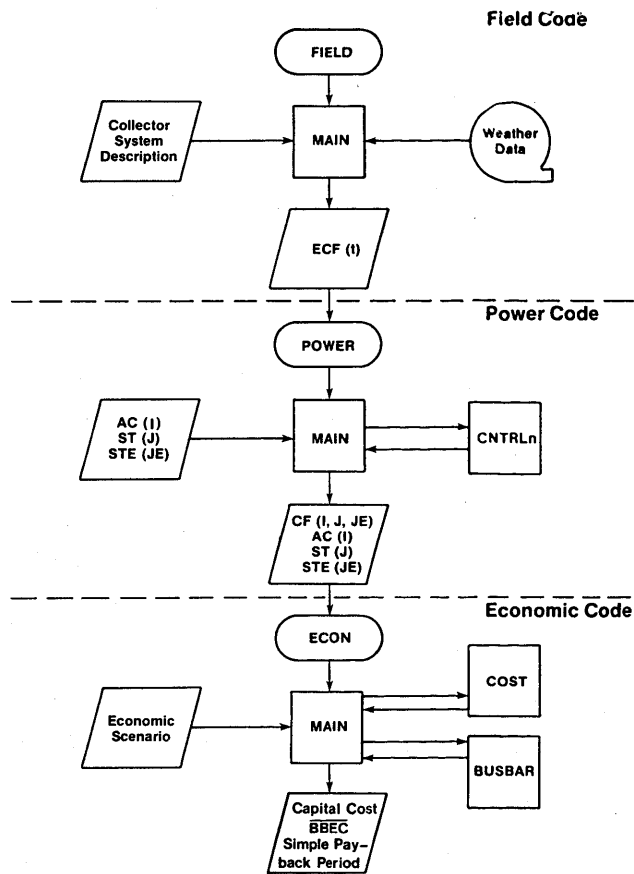


Figure 1—Simplified flow chart for BALDR-1 performance and cost simulation codes.

at each time step per unit area is once again equal to the product of the optical efficiency, direct normal insolation and the time step. The energy collected by the receiver is the energy incident on the receiver minus the thermal losses. The energy collected from the field (ECF) is equal to the product of the energy collected at the receiver and the thermal transport efficiency.

For low concentration non-tracking systems (LCNT), e.g., CPC collector, and shallow solar ponds (SSP), the total collector efficiency is determined from a linear relationship between total efficiency and  $\Delta T$  ( $T_{\text{collector}} - T_{\text{ambient}}$ ). These relationships were based on plots of test data for advanced concept versions for each of the two collector types. (The y-intercept,  $\Delta T=0$ , is equal to the optical efficiency.) The energy collected from the field (ECF) is equal to the product of the total collector efficiency (including thermal losses), insolation, the time step, and the thermal transport efficiency. For the LCNT, insolation was taken as the sum of direct normal plus diffuse divided by the concentration ratio. For the SSP, insolation was taken as direct normal plus diffuse, or global.

The variables passed to the POWER code include an array of values of ECF for each time step, dry-bulb and wet-bulb temperatures, and unit collector area.

## POWER CODE

The POWER code is a second-order simulation based on the Aerospace computer code as modified by JPL<sup>2</sup> and Battelle PNL.<sup>3</sup> POWER differs from the earlier codes primarily in that it provides the option of using different control algorithms for both the operation of power conversion equipment and the dispatch of electrical and thermal storage. There are currently two operational control algorithms: CNTRL2 and CNTRL3.

CNTRL2 models systems with storage of receiver fluid (e.g., salt, sodium, etc.) at approximately the same condition as it leaves the receiver, sometimes called series storage. CNTRL3 models systems with storage of an intermediate fluid (e.g., storage of oil for a steam receiver system). In this case, the temperature of storage is significantly below the receiver outlet temperature and a dual admission turbine is therefore modelled.

Both control algorithms share the following features not usually found in second order solar thermal system simulations: (1) electrical and thermal storage may both be modelled for any power plant; (2) a weighting factor may be used to reduce the value of electricity delivered above plant rating to simulate hard or soft limits on plant output; (3) the decision of how to dispatch the energy from the collector field is made for the current time step; knowledge of future insolation is not used; (4) depletion of thermal storage is limited to the value which will assure a hot start-up the following morning; the minimum allowable amount of heat in storage is then a function of the number of hours until the next anticipated morning start-up.

In addition, CNTRL2 incorporates the possibility of overload operation of the power conversion equipment for specified periods. While not currently incorporated into CNTRL3, this capability could easily be added.

CNTRL2 operates with priority on producing and delivering electricity. Thermal storage is used only when there is insufficient energy to start the engine or when there is more energy than required to produce rated power. If electrical storage is modelled it is used for leveling the plant output curve. When the engine generator output is below plant rating, the output is supplemented by energy from electrical storage.

In CNTRL3, there are three operating strategies available: electricity priority, storage priority, and peak load priority. The electricity priority strategy is identical to that used in CNTRL2. The storage priority causes thermal storage to be charged with the engine off until storage is filled to a specified fraction. Only then is the engine turned on, and the priority reverts to generation of electricity for the remainder of the day. The peak load priority option is similar to the storage charging priority except that storage is maintained at the specified fraction until a designated peak period occurs. During the peak period, the priority reverts to generation of electricity. When the peak period is over, any heat left in storage is retained for use during the following day.

Component models in POWER were written in several levels of detail according to their impact on plant perform-

ance. The engine efficiency model is a function of hot engine temperature, cooling tower temperature, and the load at each time step. The thermal and electrical energy storage residence losses are calculated based on the amount of energy in storage at each time step. The auxiliary electrical loads are calculated based on plant capacity and actual plant output at each time step. The electrical transport efficiency is based upon electrical current flow through the transport system. The component models for thermal and electrical storage charging and discharging, the electrical generator, power conditioning, the inverter and the converter currently use a constant average efficiency. The component models may be easily increased in accuracy if necessary or desirable for a particular application.

The POWER code calculates the electricity delivered to the grid at each time step and sums it for one year. The total electrical energy delivered during the year is divided by the total electricity which would have been delivered had the plant operated at rated capacity for the entire year. This yields the plant capacity factor. This capacity factor is calculated for each plant described by an element of the three dimensional matrix of collector field sizes (AC), thermal energy storage sizes (ST), and electrical storage sizes (STE).

Matrices of the operating mode of the plant and the dispatch of electrical storage at each time step can be output. The calculated capacity factor, along with the corresponding collector field size, thermal storage size and electrical storage size, is output for use by the ECON code. In addition, the plant rated capacity and generator size are output for use by ECON.

## ECON CODE

The ECON code includes two major subroutines (COST and BUSBAR) which are based on computer codes originally written by JPL.<sup>2,7</sup> Using the output from POWER, ECON determines a capital cost, a life cycle busbar energy cost, a simple payback period, and annual operations and maintenance (O&M) costs for each plant configuration based on either the thermal energy or the electrical energy produced.

Subroutine COST uses unit costs as inputs to determine the cost streams for both capital expenditures and O&M. Capital costs are determined for each of four subsystems: (1) collector and receiver, (2) electrical and/or thermal storage, (3) power conversion, and (4) miscellaneous (including land, thermal and electrical transport, and spares and contingencies). These costs are currently distributed over the plant construction period as a uniform series of payments. With slight modifications to the code, COST could create a nonuniform cost stream.

Operations and maintenance costs are also determined in COST. Currently, O&M is a uniform stream of annual costs for each year of the plant's lifetime. In case a specific schedule of required maintenance is known, COST can be modified to produce a nonuniform O&M cost stream. Alternatively, a periodic maintenance cost could be added onto the annual O&M cost stream currently produced by COST.

Subroutine BUSBAR is based on the Utility-Owned Solar Electric Systems (USES) model, a conventional present value analysis adapted for solar electric power plants by JPL.<sup>8</sup> It calculates that busbar energy cost in constant-year dollars which will generate system-resultant revenues equal to the system-resultant costs. The inputs for BUSBAR represent two types of information: system cost data and accounting information. The cost data as currently used consist of the arrays of capital costs and O&M costs which are generated in subroutine COST. Escalation rates are input for capital and O&M in addition to the general inflation rate. BUSBAR is written to handle separate maintenance charges, fuel costs and social benefits along with their appropriate rates of escalation. The ECON code also has the capability of doing only the busbar energy calculations if a net present value cost is input.

The second group of input data, the accounting information, represents the variables that are used to determine the cost of capital. From these data, the discount rate, the fixed charge rate, and the capital recovery factor are determined in BUSBAR.

An additional capability exists within ECON for producing plots of the data generated. Subroutine PLOTIT can be called to produce a graph of busbar energy cost versus capacity factor for each system. For the systems which use either thermal or electrical storage, but not both, the graph will have a set of curves, each of which represents a distinct value of collector area with points marked representing various amounts of storage (e.g., Figure 2). For the systems which use both electrical and thermal storage, a separate plot will be generated for each value of collector area. Each plot will consist of a set of curves, each representing an amount of thermal storage with points marked representing amounts of electrical storage.

## COMPUTATIONAL TIME

To simulate the annual performance of a point focus central receiver system on a CDC 6600 computer using 15-minute time steps, approximately 50 seconds of CPU time is required for FIELD and approximately 300 seconds of CPU time for POWER for a full matrix of collector areas and storage sizes for electrical output cases. ECON requires approximately 10 seconds of CPU time in the corresponding simulation.

## SUMMARY

A system simulation has been written to model the performance of solar thermal power systems for both electrical and process heat applications. The models are modular allowing for easy use and modification. Annual performance and economics of most proposed solar thermal systems can be modelled by the simulation in its present form.



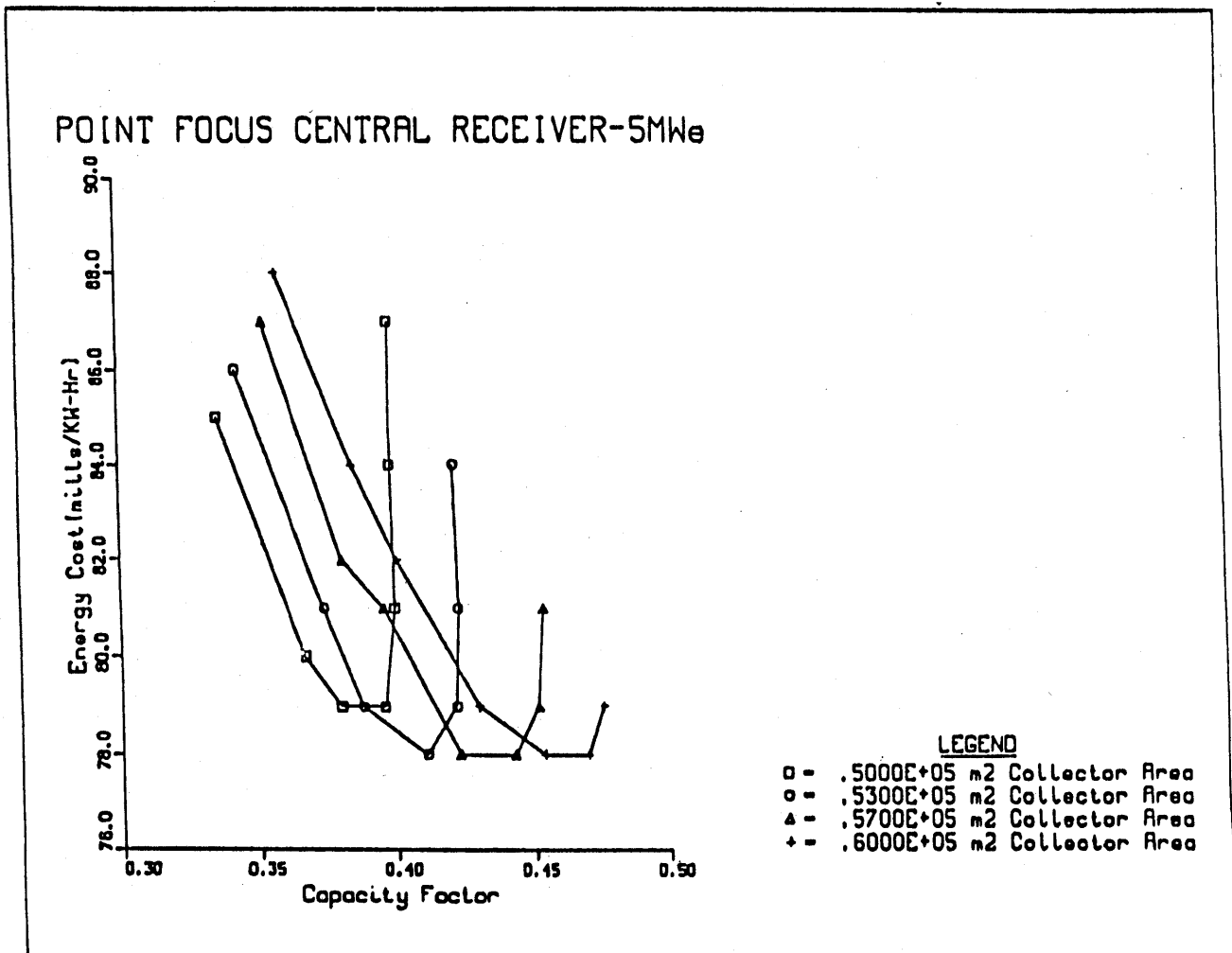


Figure 2—Graphical output from ECON: each point marked on a curve represents a different amount of storage.

#### ACKNOWLEDGMENTS

The development of this simulation is the work of many SERI employees. The following people participated in the development of BALDR-1.

M. Buhl	J. Kowalik
S. Cronin	L. Lacy
M. Edesess	D. Madison
A. Edgecombe	R. Mitchell
J. Finegold	L. Morrison
A. Herlevich	R. O'Dougherty
M. Karpuk	J. Pagano

L. Morrison and A. Edgecombe are singled out for particularly important contributions in the early stages of model development. We also gratefully acknowledge the support of J. Thornton, Task Leader of the Small Power Systems Study, under whom this work was conducted. Funding was provided under DOE Contract EG-77-C-01-4042.

#### REFERENCES

- Thornton, J., Brown, K., Edgecombe, A., Finegold, J., Herlevich, A., Kriz, T., "Comparative Ranking of 1-10MW<sub>e</sub> Solar Thermal Electric Power Systems—An Executive Overview," SERI/TR-35-238, Solar Energy Research Institute, Golden, CO, September 1979.
- El Gabalawi, N., Hill, G., Bowyer, J., Slonski, M., "A Modularized Computer Simulation Program for Solar Thermal Power Plants," JPL 5102-80, Jet Propulsion Laboratory, Pasadena, CA, July 1978.
- Bird, S., "Modification of the JPL Solar Thermal Simulation Code for Use in the PNL Small Solar Thermal Power Plant Systems Analysis," Battelle Pacific Northwest Laboratory, Richland, WA, July 29, 1978.
- Dellin, T. A. and Fish, M. J., "A User's Manual for DELSOL: A Computer Code for Calculating the Optical Performance, Field Layout, and Optical System Design for Solar Central Receiver Plants," SAND 79-8215, Sandia Laboratory, Livermore, CA, June 1979.
- Leary, P. L. and Hankins, J. D., "A User's Guide for MIRVAL—A Computer Code for Comparing Designs of Heliostat-Receiver Optics for Central Receiver Solar Power Plants," SAND 77-8280, Sandia Laboratory, Livermore, CA, February 1979.
- Dudley, V. E. and Workhoven, R. M., "Summary Report—Concentrating Solar Collector Test Results Collector Module Test Facility (CMTF) January-December 1978," SAND 78-0977, Sandia Laboratory, Albuquerque, NM, March 1979.

- 
7. Slonski, M. L., "Energy Systems Economic Analysis (ESEA) Methodology and User's Guide," JPL 5101-102, Jet Propulsion Laboratory, Pasadena, CA, February 15, 1979.
  8. Doane, J., O'Toole, R., Chamberlain, R., Bos, P. and Maycock, P., "The Cost of Energy from Utility-Owned Solar Electric Systems: A Required Revenue Methodology for ERDA/EPRI Evaluations," JPL 5040-29, Jet Propulsion Laboratory, Pasadena, CA, June 1976.



# Overview of the alternative power system economic analysis model

by RICHARD B. DAVIS

*Jet Propulsion Laboratory*  
Pasadena, California

and

JEROME V. V. KASPER

*UCLA*  
Los Angeles, California

## INTRODUCTION

The Alternative Power System Economic Analysis Model (APSEAM) is an interactive computer model which can be applied in three ways: (1) the model projects the annual, after-tax costs of capital investment in various conventional and non-conventional energy technologies for each and every year in the investment time horizon; (2) the model serves as an investment analysis tool; and (3) the model serves as a tool which can be used for evaluating the impact of specific policies on specific investors.

The basic model premise is that the valuation of investment alternatives should have a "lifecycle cost" perspective in addition to the usual "first-cost" perspective. The needed "lifecycle cost" perspective is obtained through use of a cash flow methodology. Detailed cash flow information is projected for each investment alternative for each and every year in the investment time horizon. This annual cash flow information is then aggregated to produce various measures of the lifecycle costs of each of the investment alternatives. The model can be used to quantify the impact on annual cash flows and on lifecycle costs of variations in technology cost (capital costs and operations and maintenance costs), general economic conditions, investor-specific financial conditions, the method of financing of the capital investment, the resource (e.g., solar insolation, geothermal well, etc.), technology performance over time, supply and demand matching, incremental plant start-up, and component replacement scenarios.

As an investment analysis tool, the model aggregates the projected cash flow information to produce an investor-specific "investment profile" for each investment alternative—a set of figures of merit which include net present value, levelized energy costs, liquidity requirements, and fractional return on investment. As an investment analysis tool, the model produces investor- and application-specific projections of how specific investors are likely to perceive the

worth of a particular investment alternative relative to others.

The specific investor types which can be treated include private utilities, municipal utilities, corporations, and individuals. In addition, various types of joint ventures and leasing arrangements can be modeled.

As a tool for analyzing policies, the model quantifies the impact of specific state and federal actions on the perception of specific private sector investors concerning the economic viability of the various investment alternatives. For example, the model can quantify the implications of utilizing various methods of depreciation accounting, various provisions for tax credits, various rules concerning the carry-back and carry-forward of tax credits and/or operating losses. Insofar as the model is a year-specific cash flow model, these governmental actions can be year-specific.

The model has been applied to a study of the economics of a homeowner in Phoenix either buying or leasing a photovoltaic system rather than relying on the utility grid for all electricity needs. The model is currently being applied to evaluate the financial viability of solar thermal technology in an island utility system (Catalina) and in a thermal stimulation scheme for heavy oil recovery (Kern County).

### *Cash flow model*

The basic model premise is that the evaluation of investment alternatives should be based upon a "lifecycle cost" perspective. The relative worth of the various investment alternatives in conventional and non-conventional energy technologies is particularly difficult to judge when the various cost elements associated with the investment alternatives change at varying rates over the time horizon of interest. In general, the costs of the various investment alternatives over the entire time horizon of interest must be recognized. The needed "lifecycle cost" perspective is obtained through use of a cash flow methodology. In a cash

flow model, detailed cash flow information is projected for each investment alternative for each year in the investment time horizon. Within APSEAM, this annual cash flow information is aggregated to produce various measures of the lifecycle costs of each of the investment alternatives.

The model is capable of quantifying the effects of variations in the cost of various technologies (capital costs and operations and maintenance costs), general economic conditions, investor-specific financial conditions, the method of financing of the capital investment, the resource (e.g., solar insolation levels), technology performance over time, supply and demand matching, incremental plant start-up, and component replacement scenarios.

### Investment analysis tool

The Alternative Power System Economic Analysis Model also functions as an investment analysis tool. As such, it seeks to answer the question: "What is the relative worth of different investment alternatives to a specific investor?" This question is much broader than the question "What are the lifecycle costs\* of different investment alternatives?" for it takes into account a specific investor's financial environment (for example, his ability to take advantage of tax credits, his cost of capital, etc.) as well as the specific investment alternatives available to that investor. As applied to energy system investments, the investment alternatives can include: (a) capital investments in various energy technologies (conventional or non-conventional) to meet specified energy requirements (electrical and/or thermal); or (b) purchase of all energy needs (electrical energy from the utility grid, thermal energy from combustion of purchased fossil fuel in fossil-fired boilers); or (c) cessation of those activities which create energy needs—investment in some alternative with no energy demands.\*\*

The model "massages" the projected cash flow information and then aggregates it to produce an investor-specific "investment profile" for each investment alternative—a set of figures of merit which enable that investor to make an informed decision.

### Investor-specific policy analysis tool

In addition to functioning as a lifecycle cost model and as an investment analysis tool, the model also functions as a policy analysis tool. As such, it seeks to answer the question: "What is the impact of various governmental policies on specific private sector investors concerning the relative worth of various investment alternatives?"

\* It is of course true that the calculation of the lifecycle costs of an investment alternative requires assumptions about the investor's opportunity cost of investment and his state and federal tax rates.

\*\* This last investment alternative is included to reflect the fact that, all other things being equal, private sector investors seek to maximize profit. If being in a business which produces or consumes energy is not as profitable as being in another type of business, an investor might be expected not to opt for investments involving energy production or consumption.

The model quantifies the impact of specific state and federal policies on the perception of specific private sector investors concerning the economic viability of the various investment alternatives. For example, the model can quantify the implications of utilizing various methods of depreciation accounting, various provisions for tax credits, various rules concerning the carry-back and carry-forward of tax credits and/or operating losses. Insofar as the model is a year-specific cash flow model, many of these governmental policies can be year-specific. Thus, time-phased incentive strategies can be evaluated. This is an important feature of the model insofar as the government will likely use time-phased strategies to encourage the use of alternative, non-conventional energy systems: large incentives in the near-term, with a tapering off of the incentive size as the desired energy technologies penetrate the marketplace by natural mechanisms.

### Ownership options

The specific investor types which can be treated include private utilities, municipal utilities, corporations, and individuals. In addition, various types of joint ventures and leasing arrangements can be evaluated.

### Alternative uses of model output

As an investment analysis tool, the model produces projections of how specific investors are likely to perceive the worth of a particular investment alternative in a specific application relative to other investment alternatives. This information, coupled with the market size potential which those specific investors represent, provides the basis for meaningful estimates of *expected* market penetration.† Hence, model-derived information can serve as valuable input to macro-market penetration models. In like manner, as a policy analysis tool, the model specifies what the impact of specific policy decisions is on the perceptions of specific investors in specific applications concerning the relative worth of various investment alternatives. Aggregated, this information enables the effects of alternative governmental policies and incentive strategies on the market penetration potential of various energy technologies to be quantified. In this way, the costs and the expected benefits associated with alternative policy options can be related and optimal trade-offs identified, both from the standpoint of the government and of individual investors.

### Model execution

In actual operation, the model is a highly interactive program which possesses graphical capabilities. Internal doc-

† Actual market penetration will depend on largely subjective behavioral considerations and institutional/political factors as well as upon the purely financial, profit-maximizing decision-making criterion specified here.

umentation is provided in the program. It has many user-oriented features to allow for easy input/output formatting, internal calculations and sensitivity analyses. In addition, its graphical capabilities enable the user to readily perceive trends and relationships.

## OVERVIEW OF MODEL METHODOLOGY AND OPERATIONAL CAPABILITIES

The model compares each capital investment alternative to a "business-as-usual" option. This "business-as-usual" option is the status quo.

The energy production systems associated with each capital investment alternative are designed so that a uniform degree of energy production reliability<sup>††</sup> is realized by the investor in his application. In this way, the various investment alternatives can be directly compared in terms of the financial aspects alone.

The comparison of a specific capital investment alternative to the "business-as-usual" option is realized by: (1) projecting the expected annual after-tax cash flows associated with that investment alternative; (2) projecting the expected annual after-tax cash flows associated with the "business-as-usual" option; and (3) calculating the annual differential after-tax cash flows.

The input data required to effect this comparison falls into one or more of the following categories:

- Purchased power costs (PP)
- Capital investment costs (CI)
- Capital funding costs (CF)
- Recurrent expenditures (EX)
- Revenues (RV)
- Taxes (TX)
- Escalation rates (ER)
- Economic parameters (EP)
- Performance (PR)
- Demand (DE)
- Timing (TI)

The major steps in the process are as follows:

- 1) The matching of supply (the performance of the Capital Investment Alternative) and the demand (the Load Profile)
- 2) Determination of the pre-tax cash flows for the Capital Investment Alternative
- 3) Determination of pre-tax cash flows for the Business-As-Usual Option
- 4) Determination of the Subsystem Replacement Requirements

<sup>††</sup> The term "reliability" here refers to the probability that a power system will be able to satisfy the investor's demand for energy. Uniform reliability implies that the power system configuration which constitutes each of the investment alternatives has the same loss of load probability (as expressed in terms of the number of days within a given time period that the demand cannot be satisfied). Hence, for example, sufficient back-up or standby capacity is provided in the definition of an insulation-dependent technology alternative so that its reliability is equal to that associated with all other investment alternatives and with the "business-as-usual" alternative.

- 5) Determination of state and federal taxes for the Capital Investment Alternative and for the Business-As-Usual Option

- 6) The calculation of the After-Tax Differential Cash Flows.

## MODEL APPLICATION

### *Lifecycle costing*

APSEAM is a cash flow model which projects, for each investment alternative considered, all of the revenues and expenses to be experienced by an investor over the entire investment time horizon. These annual cash flow streams are summed to determine the lifecycle costs for each investment alternative.\* Four different measures of lifecycle costs are calculated, corresponding to the four different types of "dollars" which the model considers. These four types of "dollars" are: (1) nominal dollars; (2) real dollars; (3) energy dollars; and (4) discounted dollars.

*Nominal* dollars are dollars actually "in hand." In inflationary times, they possess less and less purchasing power over time. *Real* dollars are nominal dollars from which the effects of inflation have been decoupled. Real dollars have constant purchasing power over time for the "general mix" of consumer goods. *Energy* dollars are real dollars in which the effects of inflation and of a real escalation of energy costs have been decoupled. When energy costs escalate at a rate above the general inflation rate, a constant number of *real* dollars possesses less and less purchasing power over time for "energy goods." Hence, there is a need to work with "energy dollars": dollars of constant energy purchasing power over time. *Discounted* dollars are nominal dollars which have been "interpreted" to reflect the time value of money to a specific investor and to account for the alternative uses he might make of his investment dollars. Assume that an investor has an opportunity cost of investment, in nominal after-tax dollars, of OCIN.\*\* This is that investor's time value of money, his "discount rate." If that investor were to receive  $A$  after-tax dollars one year from the present, he would be indifferent between that and receiving  $A/(1+OCIN)$  dollars today; therefore the present worth of a cash flow,  $A$ , one year in the future, is  $A/(1+OCIN)$ . In general, for the cash flow stream,  $C_t$ , and an investor discount rate, OCIN, the present worth of that cash flow stream extending over  $N$  years is:

$$\text{present worth} = \sum_{t=1}^N \frac{C_t}{(1+OCIN)^t}$$

Assuming no risk, an investor is indifferent between receiving the actual cash flow stream over time or having the pres-

\* Normal model operation produces differential cash flow information, the cost of a capital investment option relative to the cost of the "business-as-usual" or "purchased power" option. To obtain lifecycle costs, the cost of the purchased power option is set to zero.

\*\* The opportunity cost of capital contains an opportunity cost component and a risk premium component.

ent worth of that cash flow stream today—they are financially equivalent.

The lifecycle cost for each investment alternative is calculated in terms of these four types of dollars by expressing the after-tax cash flow stream for each investment alternative valued in each type of dollar and by then summing each of them.

The *discounted* life cycle cost of an investment alternative is perhaps the most informative of the four different types of lifecycle costs to a potential investor. The discounted lifecycle cost specifies the effective amount of money the investor must have on hand, today, invested at his discount rate, OCIN, so that he can "cover" the costs associated with that investment alternative as they come due over the investment time horizon.

### INVESTMENT ANALYSIS TOOL

The information contained in the annual differential cash flows must be aggregated in order for a potential investor to more readily evaluate the relative worth of the various investment alternatives. In acting as an investment analysis tool, the model aggregates the annual differential cash flow data to produce an "investment profile." This investment profile contains the type of information which enables a decision maker to make an informed decision. This investment profile contains five different figures of merit, each of which is expressed in terms of one or more of the four different types of "dollars" described above.

These figures of merit are:

- 1) Net Present Value
- 2) Absolute, Levelized Energy Costs (after-tax)
- 3) Liquidity Requirements
- 4) Fractional Return on Investment
- 5) Payback Period.

### POLICY ANALYSIS TOOL

The model can be used to determine the impact of certain types of governmental actions on the cash flow stream and resultant figures of merit associated with the alternative investment choices by specific investors. A series of options exist in the model which the user can specify when executing the program.

These options were chosen insofar as they represented various means whereby the government could alter the perceptions of the private sector concerning the economic viability of various investment alternatives. These options fall into the following major categories:

- Depreciation accounting
- Tax credit accounting
- Income tax brackets
- Income tax deductions accounting
- Taxable income
- Property taxes

In the *depreciation accounting* category, options exist both as to method and application. The user must specify the depreciation method to be used at the federal and state level for initial investments as well as for replacements. The methods available are: (1) sum of the years digits; (2) straight line; (3) double declining balance; and (4) declining balance at a user-specified rate. The user must also specify if depreciation is to be taken at the federal and/or at the state level and, if so, on what fraction of capital costs. Finally, the user must also specify if the special federal first year 20 percent depreciation allowance is to be taken.

In the *tax credit accounting* category, options exist both as to method and application. The user must specify the terms of any tax credit, the size of the tax credit (usually a percentage of the capital cost) and the limit on the monetary amount of the credit (for example, the federal energy tax credit is 30 percent of the first \$2000 and 20 percent of the next \$8000 for a limit of \$2200). The user must also specify whether the tax credit is to be taken at the federal level and/or at the state level. Finally, if applicable, the user must specify the year the availability of the tax credit ends.

Also included in this category are options with respect to the carryback and carryover of unused investment tax credit. Presently, at the federal level,<sup>†</sup> any part of the investment tax credit which is not applied as a credit, because of the limitations with regard to the maximum credit in any given year, can be carried back three years and carried over seven years. These time periods can be adjusted in the model, to allow for evaluation of the impact of variations in them on specific investors.

Two options exist with respect to the *income tax bracket* category: the income tax brackets can be assumed to remain fixed (the status quo) or they can be assumed to be indexed, that is, to inflate at the standard rate of inflation.

In the *income tax deductions* category, options exist which are pertinent to both businesses and homeowners. With respect to businesses, the user can specify if fuel costs should be allowed as a tax deduction or not. Presently, they are, of course. The model allows this deduction to be either allowed or disallowed. With respect to homeowners, the user can specify that costs associated with buying and operating a power system are deductible, in addition to the usual interest costs. Homeowner costs, some fraction of which can be allowed as deductions at the user's discretion, include fixed costs, variable costs, other annual costs, and insurance costs. These deductions must be specified for both the state and federal levels. The rationale for allowing these options with respect to homeowners will become evident when the taxable income category is discussed.

When a company has more deductions than gross taxable income in a given year, it has a net operating loss in that year; these net operating losses can be carried back three years and carried over seven years at the federal level.<sup>††</sup> Within the model, the time limits allowed for carryback or carryover of net operating losses are input parameters.

<sup>†</sup> The State of California has no provision for investment tax credits.

<sup>††</sup> The State of California has no provision for carryback or carryover of net operating losses.

The options which exist in the *taxable income category* apply to homeowners.\* They were incorporated in the model because homeowners purchasing power systems will frequently have excess electricity to sell back to the grid and will thereby realize revenues. What is the impact if these revenues are taxed at either the federal or state levels? The user must specify, for both the state and federal levels, what fraction of those sell-back revenues are taxed. In conjunction with this, as mentioned above, the user must specify what portion of homeowner costs are deductions at both the federal and state levels against those revenues.

The *property tax option* category requires the user to specify the tax rate base on both land and improvements, the escalation rate of that tax rate base, and the property tax rates on the land tax base and on the improvements tax base. Thus, property taxes can be realized only on capital improvements, only on new land acquisitions, on both, or on neither.

## SUMMARY

The Alternative Power System Economic Analysis Model has served as the principal analytic tool for three case studies: "The Effects of Ownership Options, Government Policies, and Operational Alternatives on the Economic Viability of Residential Photovoltaic Systems," "An Analysis of the Economic Viability of a Solar Thermal Point Focusing Electric Plant for Santa Catalina Island—A Case Study," and "The Economic Viability of a Solar Thermal Plant for Enhanced Oil Recovery—A Case Study."

Two major uses of these case study analyses are:

(1) Determination of appropriate inputs for market penetration studies of various advanced energy systems: market penetration models for energy technologies typically assume that investment occurs when the levelized energy cost for the new investment alternative (such as solar) is less than or equal to the levelized energy cost from conventional alternatives (such as purchased power, oil-fired energy generation, etc.). This assumption ignores the many other financial considerations/figures of merit which an actual investor would consider in making a capital investment de-

cision. The alternative approach is to consider a particular investor/application whose characteristics are typical of a particular segment of the market. A case study for that particular investor/application is then prepared, generating the appropriate figures of merit for capital investment decision-making, and evaluating the financial feasibility of any particular investment decision. Correlation of the predicted investment choices of particular application/investor sets with the market share which each represents then provides a good basis for estimation of the market penetration of new energy technologies.

(2) Identification of optimal candidates for government-sponsored technical experiments and demonstrations. Due to the highly visible nature of demonstrations and the importance of creating a positive image of new energy systems, it is imperative that government-sponsored experiments and demonstrations be sited where the new energy systems are projected to be perceived as financially favorable as possible.

Model development is continuing. The process of applying the model to various case studies has served to identify areas of necessary/desirable model refinement. The continued application of the model to other case studies will undoubtedly identify additional model development needs.

Validation of the model has been initiated and is proceeding in two phases. Phase I is in process and involves the review of the model code/documentation by a Certified Public Accountant (CPA). Suggested/required changes are being incorporated into the model. Phase II of the validation process involves a review of the model by an accounting firm, selected through the competitive procurement process. This contract will be initiated in mid-1980.

It is anticipated that the model will be available for public use within a year. A user's manual is in preparation. The model is presently on a DEC VAX-11/780 system. It will soon be modified to be usable on an IBM 370/3032 and a UNIVAC 1108.

## ACKNOWLEDGMENT

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the U.S. Department of Energy through an agreement with NASA.

\* These are inappropriate for a business, for which these costs are already acceptable deductions.





# Computer simulation of the operations of utility grid connected photovoltaic power plants\*

by CHESTER S. BORDEN

California Institute of Technology  
Pasadena, California

## INTRODUCTION

In order to evaluate the commercial viability of photovoltaic power systems it is necessary to have reliable estimates and descriptions of the supply of electricity generated by the solar technology, the demand for that electricity, and the market application. A methodology which produces information on performance, cost, and value components of utility grid connected photovoltaic power plants has been developed to assist in these evaluations (References 1,2). This report describes that analytical model and presents an application to a utility grid connected central power system with a range of operations alternatives.

The Lifetime Cost and Performance (LCP) model is designed to causally relate the effect of hourly weather conditions and component efficiencies, system design, electrical design, long run effects of exposure (module power output degradation over time, module and balance of system failures over time, and dirt accumulation), and alternative operations/maintenance policies (timing and quantity of modules to be replaced due to failure or degradation below certain performance levels, module cleaning frequency, and balance of system operations/maintenance alternatives) to system performance, cost, and value over the photovoltaic power plant's operating lifetime. When coupled with an economic model for electric utilities, LCP provides a consistent basis for describing and comparing alternative utility grid connected photovoltaic power system designs and operating strategies. The LCP model is currently being used by the Photovoltaic Technology Development and Applications Lead Center and Low Cost Solar Array Projects at the Jet Propulsion Laboratory for photovoltaic system evaluations. These evaluations include identifying reasonable operations, maintenance and replacement strategies for a range of photovoltaic system and module designs; performing parametric studies of the effect of alternative degradation rates, failure rates, and cleaning policies on system technical and economic performance; calculating system and subsys-

tem breakeven costs; and investigating the impact of regional differences on performance, cost, value, and system design preference.

## APPROACH

The LCP model performs four operations: it

1. incorporates initial design and construction of the power plant,
2. simulates hourly photovoltaic system power output,
3. analyzes the long term effects of exposure to an outdoor environment and operations/maintenance policies, and
4. calculates the system-resultant benefits and costs associated with owning and operating a photovoltaic power plant.

Step 1 (for the central station application) begins with the initial expenditures on architect/engineer (A/E) activities including site selection, power plant design, engineering, environmental reporting, permit/licensing acquisition, site preparation, and procurement of modules (the price of which can be determined in Reference 3) and balance of system hardware. Facility construction, component installation, power plant start up and testing procedures are then specified. An incremental startup schedule for bringing capacity on-line prior to complete power plant installation can be identified by the user. The chosen installation process should reflect internal optimizations made by the A/E firm and the utility to identify potential bottlenecks and a tradeoff between the timing and size of capacity increments, costs, and worth of early utility revenues.

Step 2 is an hourly simulation of photovoltaics-generated electricity. Power output from any utility grid connected solar electricity generating facility must be calculated over a short time-frame (e.g., hourly) for a number of reasons. The primary reason is that the value to the utility of power generation (based on the coincidence of peak insolation and peak demand for electricity, the mix of power plants in the utility grid, and the cost of energy generation combined with

\* The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the U.S. Department of Energy through an agreement with NASA.

the dispatching of power plants) varies by time of day. An hourly simulation is the best way to capture the time-dependencies of photovoltaic system performance and value. Components of the analyses incorporated in Step 2 are: the basic hourly geometry (including the effects of array shading), the electrical design of the modules and branch circuits up to the power conditioning unit level (including the series/parallel connections and the effects of electrical mismatch), power plant system design, hourly weather conditions (from SOLMET data tapes (Reference 4)), module quality (based on age, as determined in step 3), and component efficiencies including those which vary over the day.

Step 3 includes the monthly effects of both power reduction due to exposure and power increase from operations and maintenance activities. The long term effects of exposure in an operating environment are modeled both deterministically and probabilistically as reductions in system power output. To determine changes in power output over time, LCP uses the following models.

(a) A Markov process is used to describe the time varying effects of degradation. In this analysis, a solar module probabilistically "migrates" to a lower module quality level until it is replaced due to failure, engineering constraints, or an economic replacement decision. Changes in module quality are incorporated as changes in the module short-circuit current level. Module degradation occurs, for example, due to yellowing of the encapsulant, cell cracking or encapsulant delamination.

- (b) An analysis which estimates the effects of electrical mismatch for a string of modules (of varying quality) connected in series or series/parallel is then performed.
- (c) Power losses due to module open-circuit failures (at a rate which varies depending on the age of the module) are then calculated.
- (d) The effect of dirt accumulation on the modules is to reduce the transmissibility of the encapsulant, thus reducing power output. The analysis which calculates monthly power losses due to dirt accumulation uses an exponential decay model.
- (e) Components other than modules may also fail causing power outages. Balance of system failures are treated probabilistically.
- (f) Operations and maintenance activities are both input by the user and derived by the LCP computer program. Cleaning policies, for example, are input as either a specified number of cleanings per calendar month, or continuous for a given duration. Replacement of modules occurs when there is a failure, a reliability constraint is encountered, a replacement activity is exogenously input, or it is endogenously calculated to be economically attractive to make replacements. Balance of system maintenance proceeds as specified by the user.

LCP causally relates each of these occurrences to their effect on power reduction or improvement, cost, and value on a monthly basis over the power plant lifetime.

TABLE I.—LCP model

INPUTS	LCP MODEL	OUTPUTS
Power Plant Location Latitude/Longitude Hourly Insolation (S)/Temperature Climatic Conditions	Initialize Program Variables Verify Inputs Start Simulation Calculate Initial Non Energy Parameters Financial (Fixed Charge Rate) Baseline Electrical Design Calculate Initial Energy Parameters Initial Hourly Power Output (nominal) including $\eta_{Temp}$ , $\eta_{PCU}$ , $\eta_{BOS}$ Utility Grid Analysis Monthly Calculations Module Power Degradation Electrical Mismatch in Branch Circuit } ( $\eta_{Deg}$ ) Module Failures/Replacements } ( $\eta_{Int}$ ) BOS Failures/Replacements } ( $\eta_{Int}$ ) Dirt Accumulation/Cleaning } ( $\eta_{Clean}$ ) Compute Energy Output for Month k (at PCU level) = $\sum_{\text{Branch Circuits}} \sum_{\text{Hours in Month}} \eta_{Deg} \times \eta_{Mod} \times S \times A \times \eta_{Temp} \times \eta_{PCU} \times \eta_{BOS} \times \eta_{Clean} \times \eta_{Int}$ Energy for Entire Plant = $\sum_{\text{Increments to Capacity (ITC)}} (\text{PCU Energy at Month k}) \times (\text{No. of PCUs in ITC})$ Evaluate Alternative Replacement Scenarios Input or Endogenously Calculated (Benefit/Cost Analysis) Compute Monthly Capital Costs and Expenses* Execute Economic Model	Monthly Energy Generated Monthly Capital Expenditures Monthly Expenses Amount and Value of Conventional Energy Displaced from Utility Grid Amount and Value of Conventional Capacity Displaced from Utility Grid Life Cycle Cost Net Present Value Cost per Unit Energy
Power Plant Design Electrical Configuration Array Configuration Component Efficiencies Facilities Support Equipment		
Photovoltaic Module Characteristics Short Circuit Current Open Circuit Voltage Area (A)/Efficiency ( $\eta_{Mod}$ ) Distribution of Module Quality Degradation/Failure Rates		
Plant Construction/Startup/Test		
Operations and Maintenance (O&M) Cleaning Frequency/Effectiveness Replacements/Repairs Balance of System (BOS) O&M		
Financial Attributes Capital Expenditures Over Time Expenses Over Time Salvage/Residual Value Inflation/Escalation Rates Discount Rate Tax Environment		
Utility Grid Time of Day Demand Marginal Cost of Generation Mix of Power Plants		

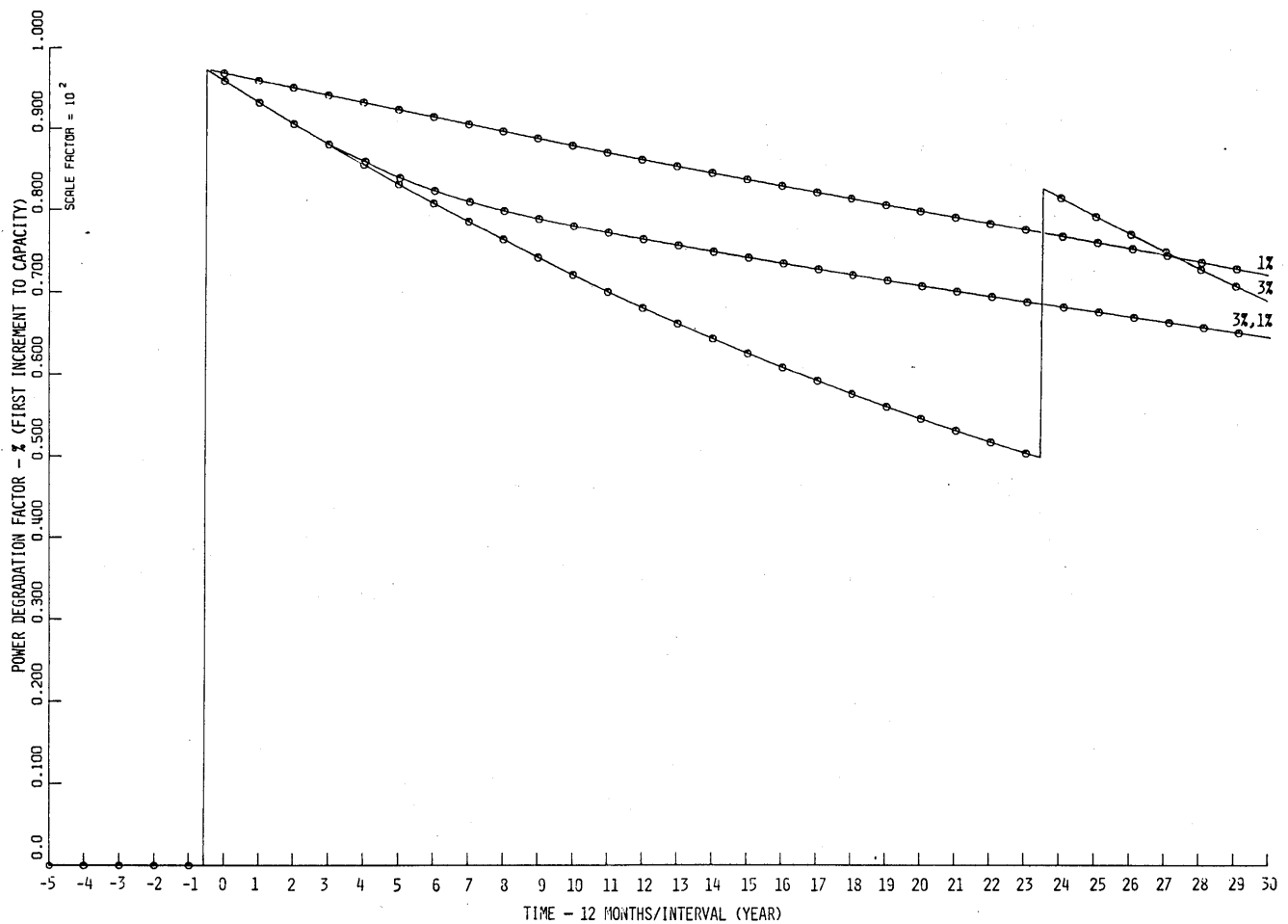


Figure 1—Power degradation over time.

LCP describes the benefits of utility ownership of photovoltaic systems in terms of the costs not incurred in generating electricity from other conventional sources (capital, fuel, and operations/maintenance costs). Step 4 of the LCP approach first determines the value of photovoltaics as a conventional energy production cost saver in terms of the short-run marginal cost (fuel and variable operations/maintenance) of conventional energy generation not spent. Time of day (hourly) utility load and marginal cost are analytically matched to the (hourly) photovoltaic system power output (see Steps 2 and 3). Savings for conventional capacity displacement are then calculated either as a reduced capacity charge by time of day included in the marginal cost of energy displaced (above), or as a separate capacity displacement calculation from any of a number of utility grid simulation models. These grid simulation models determine utility system reliability in terms of a loss of load probability constraint (References 6,7). Finally, costs for photovoltaic power plant design, construction and operation are incurred. These costs are aggregated into either capital-related expenditures or expenses (for tax purposes).

These dollar and energy flows are input to an economic

model which calculates the life cycle cost and busbar energy cost for utility-owned solar electric systems (USES) (Reference 5) and then determines the net present value of the photovoltaic system to the utility owner. Energy cost and net present value are the two decision criteria for identifying the most cost-effective alternatives. All results are necessarily location-, application-, and utility-specific.

A summary of the major inputs, routines and outputs for the model described above is shown in Table I.

#### LCP OUTPUT CAPABILITIES

The LCP model generates monthly streams of energy output, cost, and value. As discussed above, monthly energy generation combines the effect of hourly power output, the long term effects of degradation, failure and dirt accumulation, and operations/maintenance activities. Figures 1, 2 and 3 are presented to help illustrate the analytical processes and show sensitivities.

Figure 1 displays several photovoltaic module power degradation alternatives over time. Depending on the module

technical design and operating environment, a range of degradation rates may be envisioned. Based on the degradation rate, a power multiplier is calculated which is used to reduce the monthly system power output from its initial capability. These curves reflect the initial module quality distribution, all changes to that distribution over time based on the Markov process (which includes both degradation and the replacement of failed modules) and the effects of electrical mismatch. The curve labeled "3%, 1%" reflects a degradation rate which changes over the module lifetime. In this example, modules degrade at a 3 percent rate until their short-circuit current is reduced to 60 amps (from a design point of 69 amps), then the degradation rate is reduced to 1 percent. At the 3 percent annual rate, it would take the mean module 5 years to reach 60 amps. The curve "3%" degrades at that constant rate over its lifetime until the system encounters a power output constraint (at 50 percent of original capability). At that time, modules are replaced from the module quality distribution (starting with the worst short circuit current and continuing to higher quality levels) until a specified level of power improvement is achieved. Note

that the power degradation multiplier has its first non-zero amount six months (1/2 year) prior to power plant capacity generation at year = 0. This reflects the fact that the first increment to capacity comes "on-line" six months early.

Figure 2 shows power reduction over the months of the year based on the monthly power loss rates due to dirt accumulation and the alternative cleaning schedules. The curves of "Weekly" and "Monthly" cleaning schedules are not constant over a year since dirt accumulation and rain effects on power output vary during the year. In the case of no cleaning, "Rain Only," the cleansing potential of heavy rains is included for the winter months. The "Monthly" cleaning case includes the same three months of heavy rains and nine scheduled cleanings.

Figure 3 combines all the hourly power variations (from Step 2) and the monthly effects of exposure and operations/maintenance (Step 3) over the power plant lifetime. It incorporates the "3%, 1%" power degradation rate from Figure 1 and the monthly cleaning from Figure 2.

Figure 4 illustrates all of the pre-tax cost flows associated with the power plant shown in Figure 3. It shows high initial

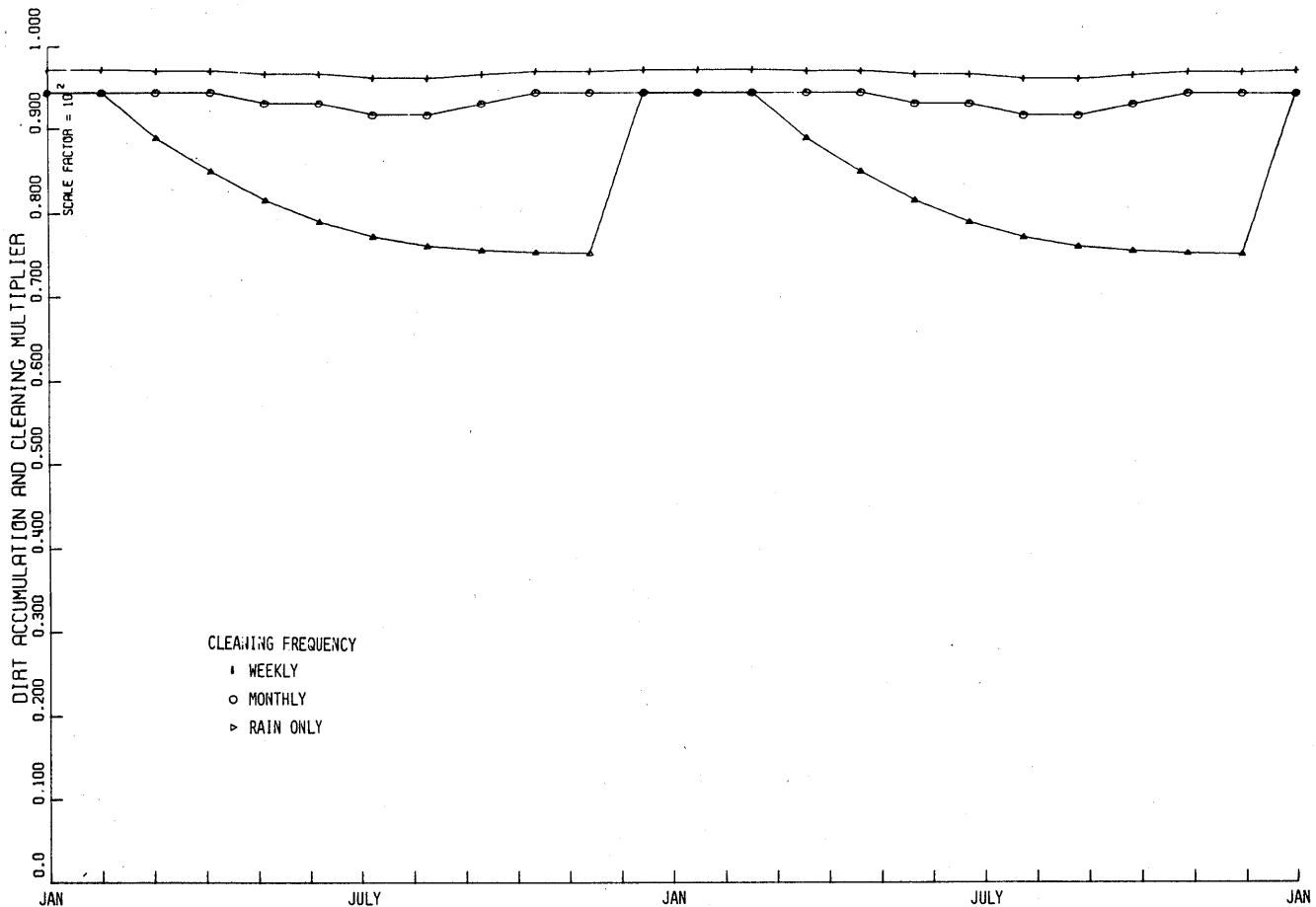


Figure 2—Dirt accumulation and cleaning effects on power output.

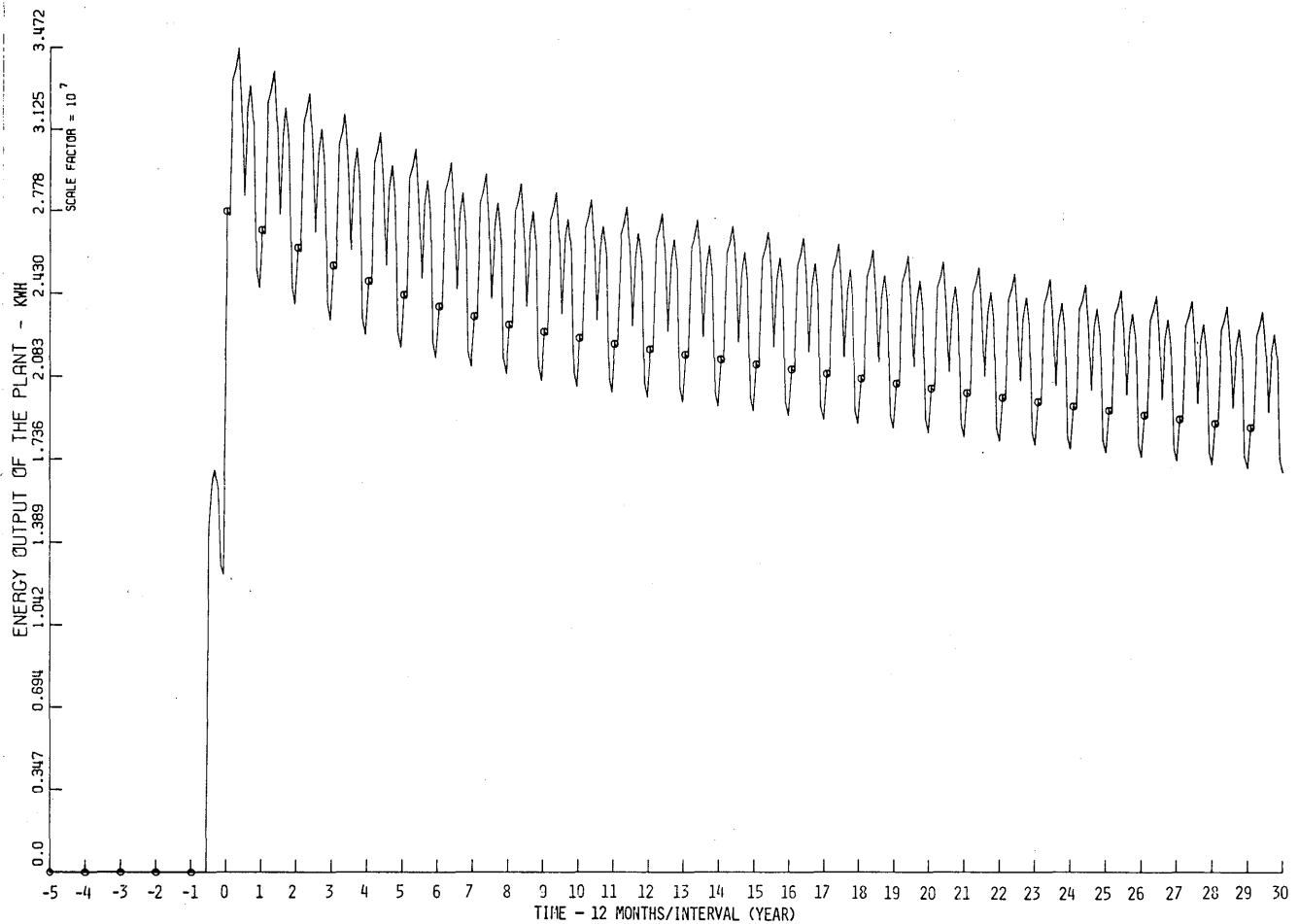


Figure 3—Energy output over time.

capital cost and relatively small operations and maintenance costs. The monthly variations during operations are due to the chosen cleaning policy, the timing of module purchases for replacement of failures, warehousing, and repair costs.

Once all of this information is generated, an economic model can then be used to determine "bottom line" results in terms of the net present value of the investment, the cost of energy, or the life cycle cost.

## STATUS

The LCP computer program is currently implemented on an IBM 370/3032 at the California Institute of Technology. It is written in SIMSCRIPT II.5, a simulation language. This model is currently being used by the Photovoltaics Program

at the Jet Propulsion Laboratory for photovoltaic system evaluations.

## ACKNOWLEDGMENTS

Development of the Lifetime Cost and Performance (LCP) Model has benefited from interactions with a number of individuals at the Jet Propulsion Laboratory. In particular, I would like to thank the following for their contributions: D. L. Schwartz (who provided many of the mathematical formulations of electricity generation) R. G. Chamberlain, Tung-Chiang Deng, J. W. Doane, P. K. Henry, and J. H. Smith. In addition, many thanks to M. C. Davisson and J. Bevan of CACI, Inc.-Federal for computer programming support.

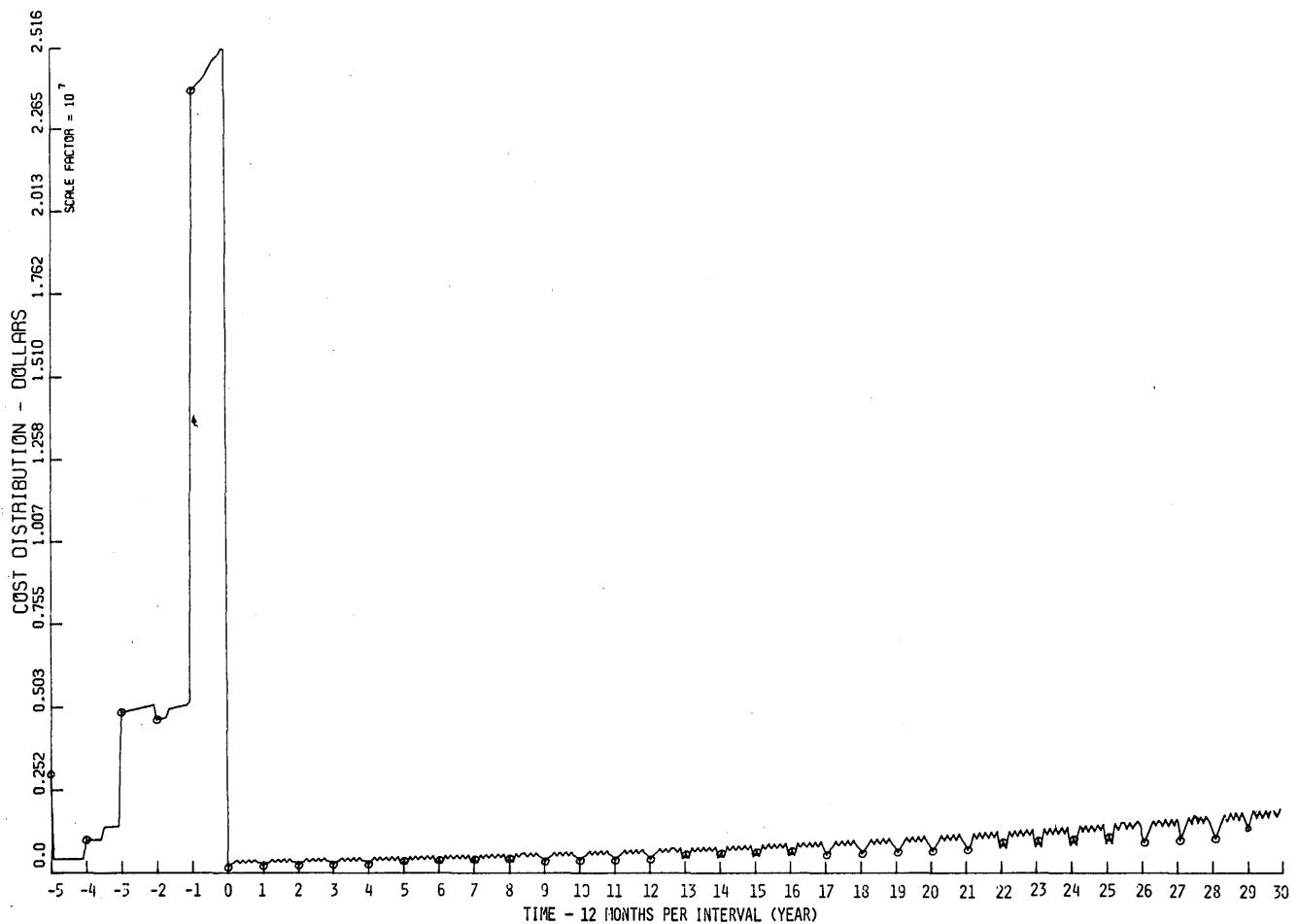


Figure 4—Expenditures over time (in nominal dollars).

## REFERENCES

1. Borden, Chester S., "Lifetime Cost and Performance Model for Photovoltaic Power Systems," *Proceedings of the 13th IEEE Photovoltaic Specialists Conference*, Washington, D.C., June 5-8, 1978.
2. Borden, Chester S. and Schwartz, Diane L., *An Evaluation Technique for Utility Connected Solar Power Systems: The LCP Model for Utility Owned Photovoltaic Systems*, Jet Propulsion Laboratory, (forthcoming).
3. Chamberlain, Robert G., *A Normative Price for a Manufactured Product: The SAMICS Methodology*, Vol. 1, Executive Summary, DOE/JPL-1012-79/5, Jet Propulsion Laboratory, Pasadena, California, January 15, 1979.
4. *Hourly Solar Radiation—Surface Meteorological Observations, SOLMET, Volume 1, User's Manual, TD9724*, U.S. Department of Commerce, National Oceanic and Atmospheric Administration, Environmental Data Service, National Climatic Center, Asheville, North Carolina, December 1977.
5. Doane, J. W., O'Toole, R. P., Chamberlain, R. G., Bos, P. B., and Maycock, P. D., *The Cost of Energy from Utility-owned Solar Electric Systems: A Required Revenue Methodology for ERDA/EPRI Evaluations*, JPL Document 5040-29, ERDA/JPL-1012-76/3, Jet Propulsion Laboratory, Pasadena, California, June 1976.
6. Finger, Susan, *Electric Power System Production Costing and Reliability Analysis Including Hydroelectric, Storage, and Time Dependent Power Plants*, MIT Energy Laboratory Technical Report No. MIT-EL-79-006, February 1979.
7. *Requirements Assessment of Photovoltaic Power Plants in Electric Utility Systems*, Report EPRI ER-685, General Electric Company for Electric Power Research Institute, Schenectady, NY, June 1978.

# Computer simulation of solar electric generating plants in a utility grid\*

by S. YOUNG, O. MERRILL, R. KNOWLES and Y. GUPTA

*Science Applications, Inc.*  
McLean, Virginia

## INTRODUCTION

Solar electric power systems have the potential to supply power for industrial, commercial, institutional, and utility applications and to reduce consumption of non-renewable fossil fuels. However, widespread utilization of solar electric technologies in the United States will require that the solar systems be operated in parallel with, or as supplements to, the existing utility grid. For such systems, assumptions regarding future electric energy costs and rate structures have a major impact on solar system design and economics. Thus, in order to fully assess the economic worth of solar electric systems, it is necessary to evaluate their impacts on utility generation characteristics and to determine solar electric system design and cost relations within the context of the overall utility/solar system interaction.

SAI has developed a methodology which evaluates the performance and economics of grid-connected solar electric technologies within the overall utility context. Because solar energy varies both hourly and seasonally, reaching a peak level for only a few hours each year, solar generation is unique relative to conventional generation currently in use by most utilities. The value of solar plants integrated in a utility network consists of both electric generation costs and capacity costs to meet a specified reliability level and depends on a number of variables: the mix and cost of conventional (non-solar) generation; the stochastic coincidence between solar generation patterns and the electric system load shape; the amount of solar penetration; the energy storage capability of the solar systems; and the solar system dispatch strategy. This paper summarizes the various techniques which have been developed and provides initial results for the worth of on-site photovoltaic, wind, and solar thermal electric technologies.

## METHODOLOGY OVERVIEW

Grid-connected solar electric systems have an impact on utility characteristics by modifying the load to be supplied by conventional generation. This provides direct economic benefits to the utility in the form of reduced fuel costs and operation and maintenance costs. In addition, the resulting load may also provide capacity savings in the form of reduced installed capacity requirements, depending on the statistical reliability of the solar generation during peak load periods; and the modified load will affect the appropriate utility generating mix of base, intermediate, and peaking plants. Figure 1 illustrates these impacts and interactions between solar electric power systems and the utility network.

The model developed by SAI provides a comprehensive analysis of the impacts of different solar electric technologies on the utility, and estimates the economic value of the solar plants to the utility, dispersed user, and/or third-party investor. The final output of the model is a set of estimates of the breakeven cost for solar electric technologies under different assumptions about ownership, payback period, and return on investment. The model calculates the economic benefits to dispersed users by assuming that the annual cost savings to the utility are passed on to the user via an appropriate rate structure. The precise nature of this rate structure is currently the subject of rather controversial legislation, partly because of the many complex interactions which are involved and which this model is designed to evaluate.

An overview of the model is shown in Figure 2(a). The overall assessment methodology involves five separate model segments: hourly simulation of solar electric system performance; utility load projection and adjustment for the output of the solar plants; capacity expansion and mix adjustment for conventional utility generation; production costing for the resulting conventional utility mix; and finally economic analysis of the solar plant value under different ownership alternatives. Because of the extensive calculations that are involved, the models have been implemented

\* This work was partially supported by the U.S. Department of Energy under Contract #XP-9-8051-1 with SERI and Contract #955238 with the Jet Propulsion Laboratory.



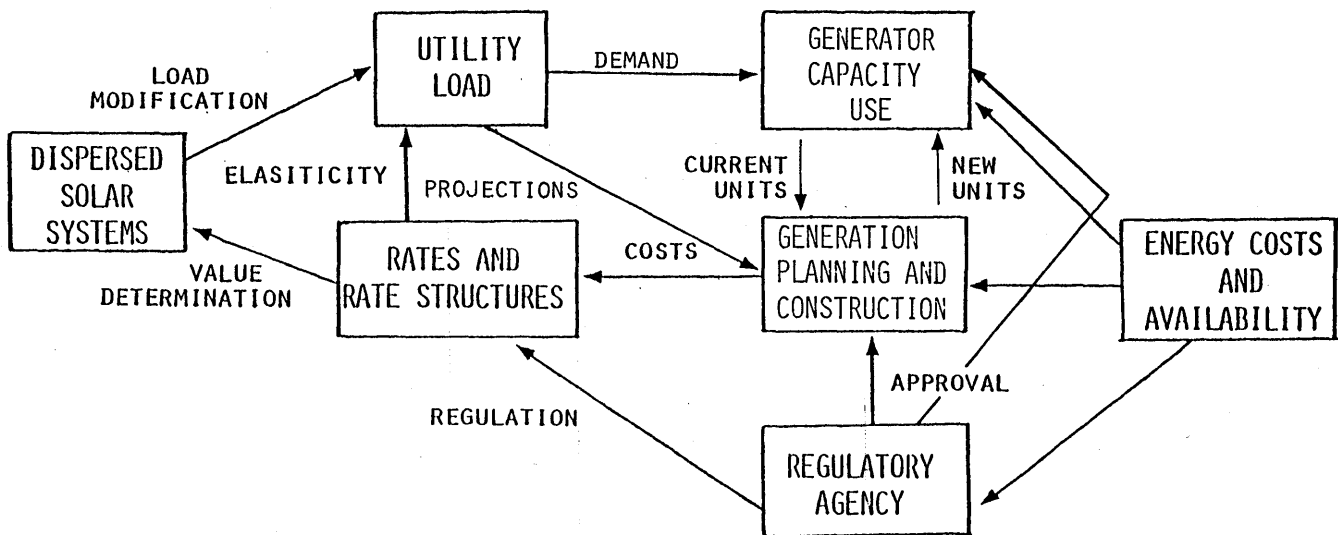


Figure 1—Solar electric power systems impacts and interactions.

with a modular structure so that analysis runs can be made independently of the others. The inputs, outputs, and analysis steps of the various model segments are summarized in Figure 2(b) and described in what follows.

#### SOLAR ELECTRIC SYSTEM PERFORMANCE MODELS

The solar electric performance models simulate the hourly output of various solar technologies. Separate models are available for photovoltaic, solar thermal electric and wind systems. Each model consists of subsystem component models which are used to compute steady-state efficiencies at each hour. As an example, Figure 3 provides an overview of the simulation model for solar thermal electric power systems. At each hour the model computes steady state energy balances, tracking losses, cosine losses, blocking and shading, reflectivity (or transmissivity), surface error losses, receiver intercept factors, receiver absorptivity, receiver re-radiation and convection losses, thermal transport losses, storage or hybrid energy flows, and part-load turbine generator efficiencies.

Inputs for the various models comprise the following categories:

- Hourly meteorological data on SOLMET tapes
  - Beam and total horizontal radiation
  - Sun position
  - Temperature
  - Wind speed
- Solar electric plant data
  - Type
  - Collector parameters
  - Energy conversion parameters
  - Subsystem efficiencies

—Storage/hybrid configuration

—Dispatch strategy

- Hourly on-site electric demand profiles

Outputs consist of the annual energy flows to/from various subsystems, overall plant performance summaries, thermal energy credits (where applicable), and hourly electric output files for total generation and energy consumed onsite. The model outputs can be used directly for systems analysis and design trade studies, or the hourly output files can be attached for input to subsequent analysis models.

#### LOAD ADJUSTMENT MODEL

The load adjustment model estimates the impact of the solar electric generation on the overall utility loads. The original loads for the utility are first projected to the time span of interest, and then the outputs of the solar electric plants are subtracted on an hourly basis, taking into account the transmission and distribution benefits of on-site generation. Solar plant outputs are scaled by the number of units and capacities of the various solar systems, and then their hourly outputs are subtracted probabilistically in the sense that various combinations of solar plant outages are considered at each hour in accordance with the forced outages probabilities. The hourly results are then accumulated in the form of load duration curves for each month or season, as indicated in Figure 4. These load duration curves are stored for both the original load projection (without solar) as well as for the solar-subtracted loads. This provides a non-solar reference case which is carried along with the solar case throughout the remaining analysis, so that the differential impacts of the solar generation can be accurately measured.

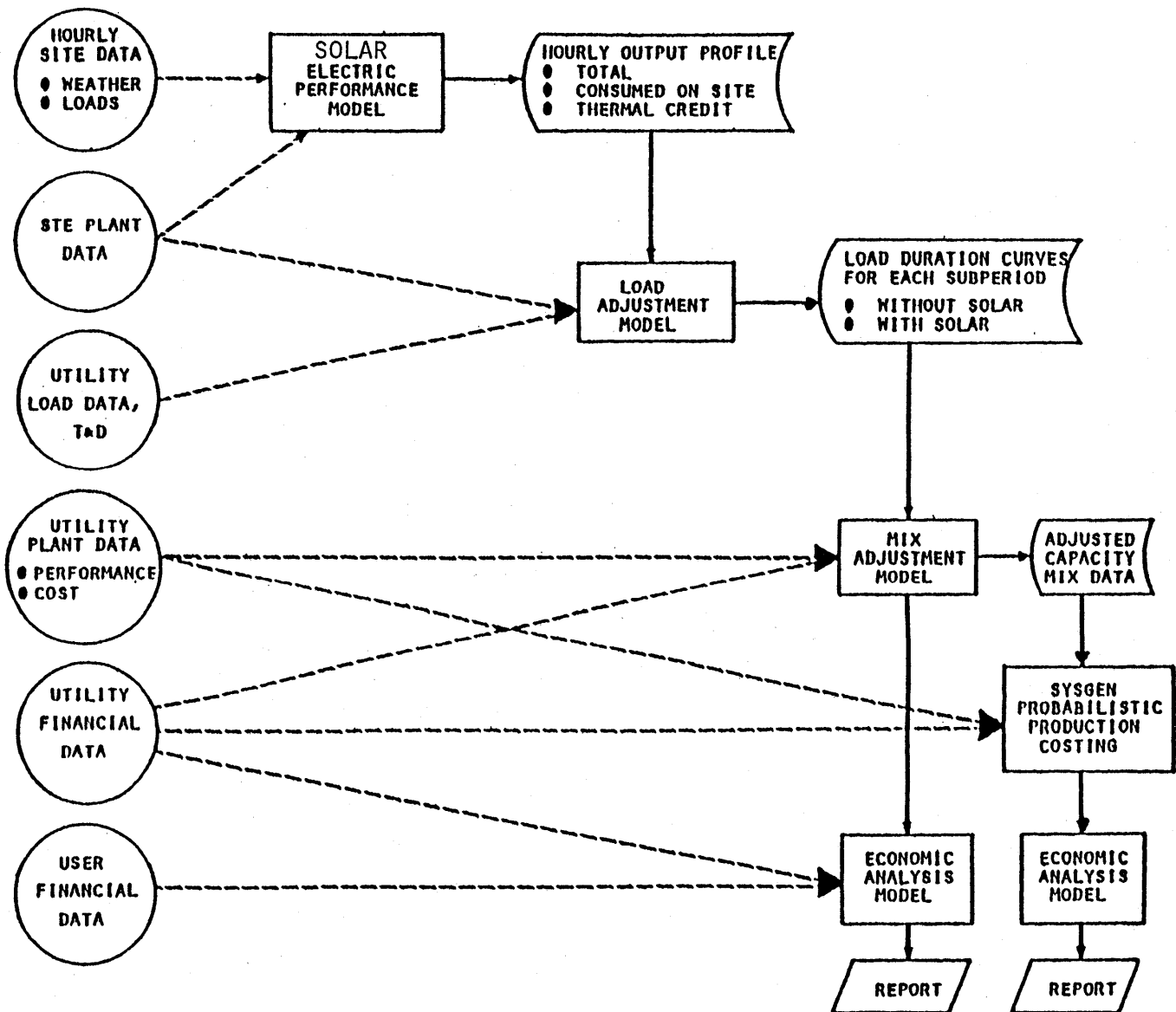


Figure 2(a)—Overview of solar electric power systems impact analysis methodology.

### MIX ADJUSTMENT MODEL

The mix adjustment model performs a capacity expansion analysis to determine the type and number of conventional generating units which should be added to the existing utility mix to meet projected electric demands at minimum total cost. This analysis is performed for both the solar case and the non-solar reference case. Inputs for the analysis include the existing utility system generating plants; the available plants for capacity expansion; characteristics of each plant type, including rated capacity, minimum operating levels, fuel type, heat rates, forced outage probabilities, maintenance requirements, fixed capital costs, and variable O&M costs; utility economic data, such as fuel costs, escalation

rates, taxes, discount rate, insurance, etc.; and projected utility load data in the form of seasonal or monthly load duration curves both with and without solar.

Figure 5 presents a screening curve analysis which illustrates the considerations involved in performing the utility mix optimization. The upper curve shows capacity costs for different plant types as a function of the number of hours per year which they are run; the lower curve represents the annual load duration curve. Capital-intensive plants such as nuclear or large coal have high fixed costs but low variable costs, so they are most appropriate when used as base-loaded plants that are run almost continuously. Combustion turbines, on the other hand, have low capital costs but high variable costs, so they are most appropriately used as peak-

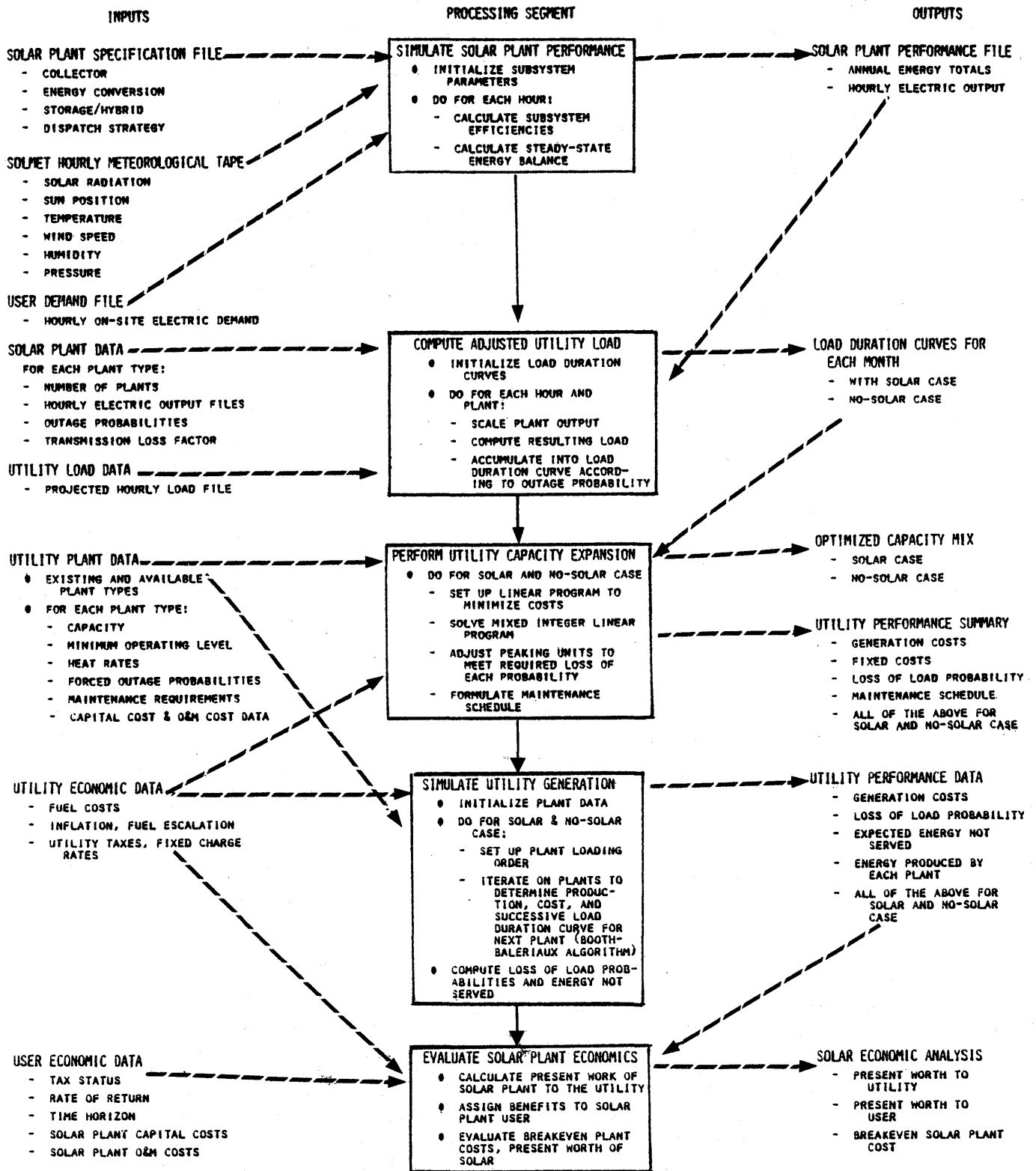


Figure 2(b)—Summary of simulation inputs, outputs, and analysis methodology.

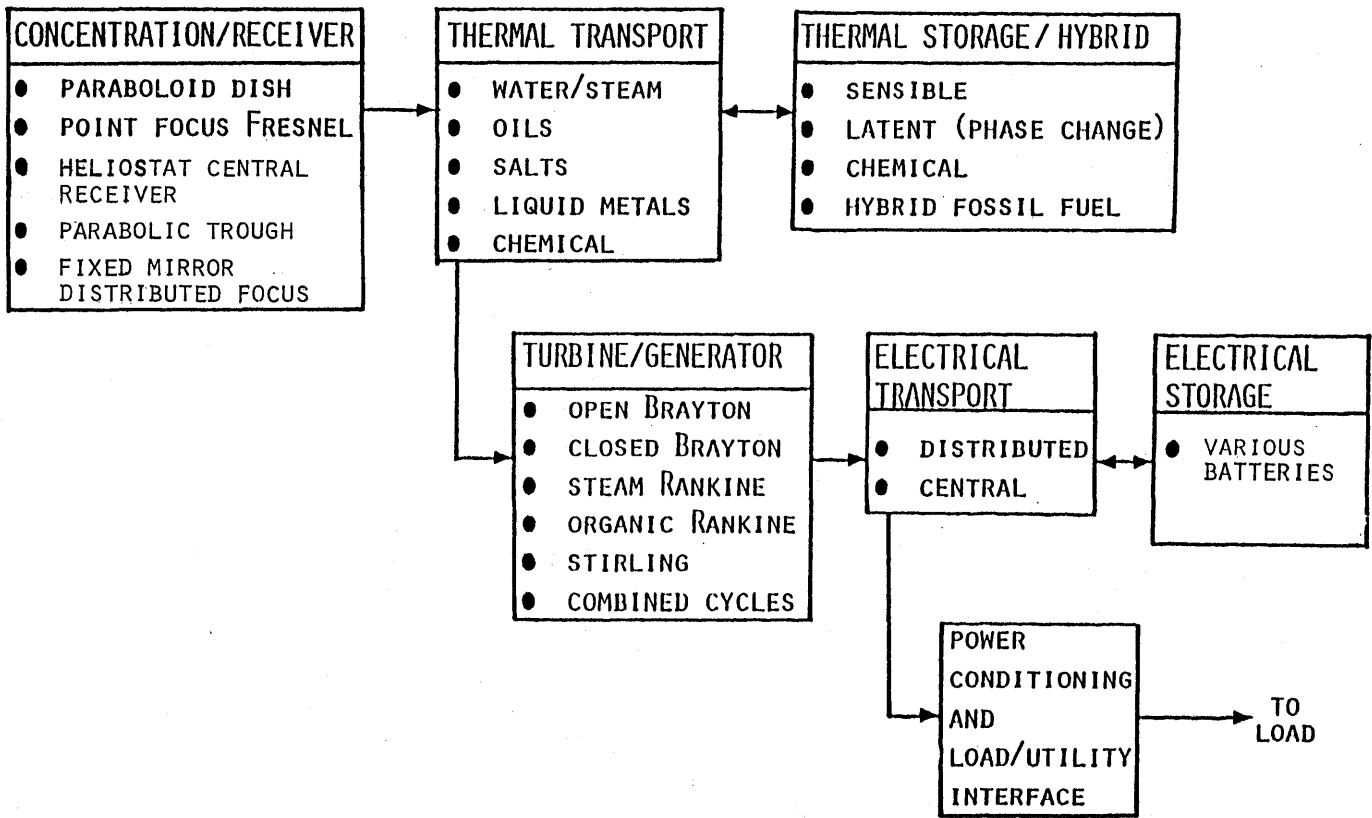


Figure 3—Solar thermal electric generating plant model.

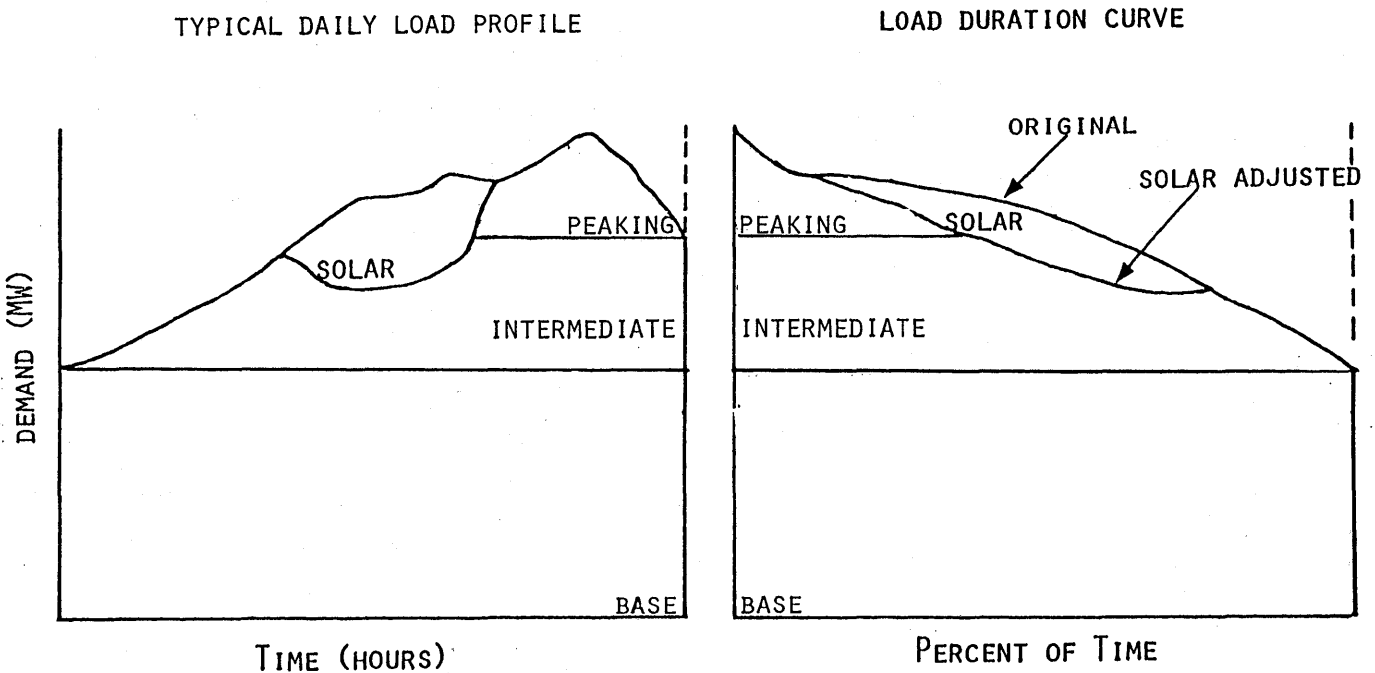


Figure 4—Formulation of utility load duration curves.

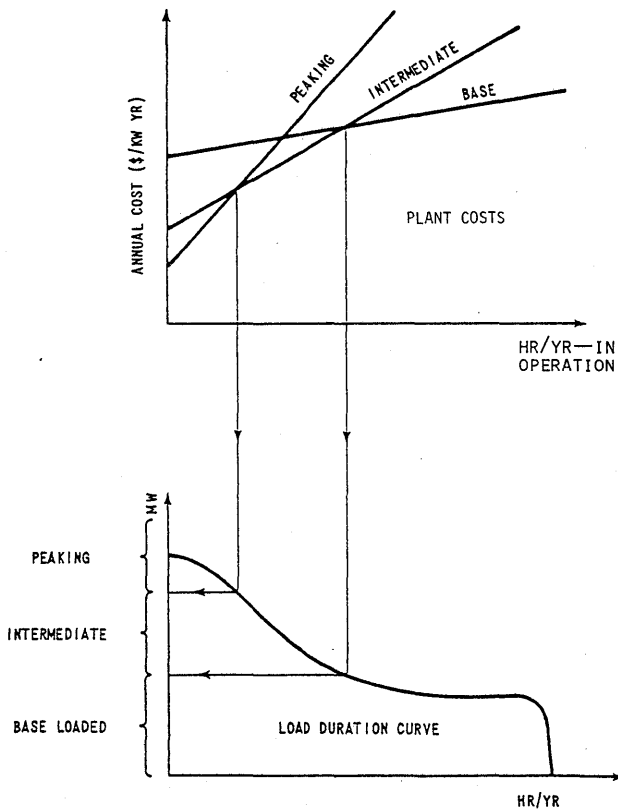


Figure 5—Screening curve analysis for mix optimization.

ing units which run only a few hours per year to meet the highest demand levels. By projecting the intersection points of the plant cost curves onto the load duration curve, as shown in the screening curve analysis of Figure 5, it is possible to estimate the amount of capacity desired for each plant type.

The screening curve analysis does not account for the previously existing plant mix of the utility, the discrete sizes of the available plants, the minimum operating levels of the plants, the spinning reserve requirements to maintain available capacity for meeting sudden load increases, or the prob-

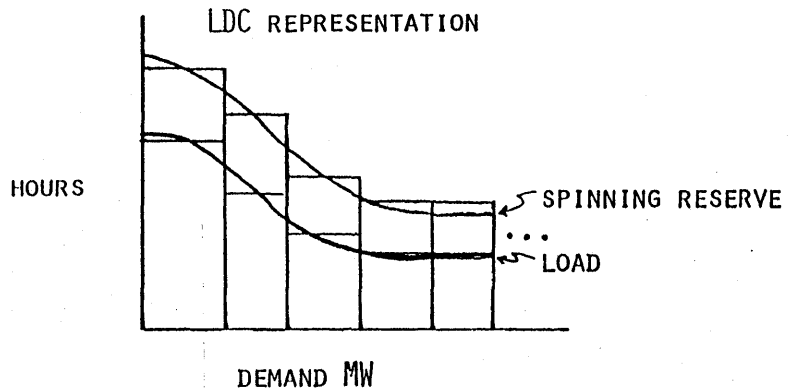
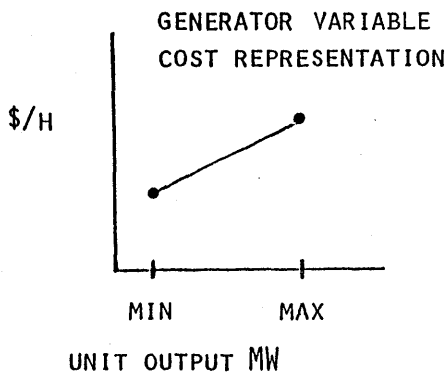


Figure 6—Linear programming formulation.

abilistic forced outage characteristics of the various plants. SAI has formulated the basis capacity expansion problem as a mixed-integer linear programming problem which is solved using a standard linear programming package with branch and bound techniques for the integer variables. Figure 6 illustrates the discretization of the load duration curve into demand segments and the variable cost representation of each generator (which allows non-linear heat rates but assumes linear incremental heat rates). The variables for the linear program are the number of plants of each type to be installed, the number of plants of each type which are dispatched in each demand segment (if minimum operating levels are accounted for), and the operating level of each plant in each demand segment. The objective function of the linear program is to minimize the present worth of total fixed plus variable plant costs. Constraints for the problem include the following categories:

- Installed Reserve Margin
- Demand Requirements
- Spinning Reserve
- Plant Capacity
- Plant Availability and Purchase Constraints
- Plant Energy Limits (e.g., Hydro)
- Integer Variable Constraints.

The solution of the linear program provides the basic capacity expansion plan; however it assumes de-rated plant capacities without accounting explicitly for the probabilistic nature of plant forced outages. This is performed in a subsequent analysis step, which estimates loss of load probability (LOLP) using a Gram Charlier series expansion technique to rapidly evaluate convolutions of the demand and plant outage random variables. Peaking capacity is then added or subtracted from the generation mix to meet the required LOLP reliability criterion. Finally, a maintenance schedule is estimated by removing plants according to maintenance requirements so as to levelize the reserve margin defined as total available plant capacity (minus peak demand) over all months. The final output of the mix adjustment model is the adjusted capacity mix (both with and

without solar), the estimated annual production costs for each generator type and fuel type, and an estimate of the present worth of revenue requirements for the utility.

#### DETAILED UTILITY PRODUCTION COSTING MODEL

A detailed probabilistic production costing model, SYSGEN<sup>1</sup>, is used if necessary to provide a refined estimate of production costs based on the modified load duration curves and the optimized conventional capacity mix for both the system with solar generation and the reference system with no solar generation. SYSGEN uses the standard Booth-Balériaux algorithm to account for plant outages, in which the effective load duration curve seen by each generator (or valve point) is expressed as the original load duration curve plus the random outages of previous generators in the loading order. The successive load duration curves are computed using a recursive technique to perform the required convolutions, as described in Reference (1).\*

\* (1) S. Finger, "SYSGEN Production Costing and Reliability Model User Documentation," M.I.T. Energy Laboratory Technical Report, May 1979.

#### ECONOMIC ANALYSIS MODEL

The outputs of either the mix adjustment model and/or the detailed production cost model are then used to provide estimates of the breakeven costs of the solar plants for utility, on-site user, and third party investor ownership alternatives. Additionally, the economic analysis can calculate the net present worth of the solar systems for various solar plant cost assumptions. The key assumption of the economic analysis is that the rate structure applied to solar system investors will reflect the difference in cost of electric service to this customer class, so that the overall savings provided by the solar plants are passed on to the investor.

#### CONCLUSIONS

The methodology described above provides a comprehensive and consistent analysis of the economic worth of different solar electric technologies operating in a utility network. This is an important consideration in determining solar electric system design and cost relations within the context of the overall utility/solar system interaction. Representative results of the modeling analysis will be presented at the conference for the worth of on-site photovoltaic, wind, and solar thermal electric technologies.



### **Numerical Methods for the '80s**

After more than a decade of emphasis on commercial applications, scientific computing is again emerging as an important area. This session will present two papers, one dealing with methods for making use of the new parallel architectures now being marketed, and the other presenting a language which makes matrix computations accessible to the scientist or casual programmer. The panel discussion will examine a number of issues of current concern in scientific computation and numerical analysis.



**Roger Firestone**  
*Area Director*





# Numerical algorithms for parallel computers

by DAVID K. STEVENSON

*Zilog, Inc.*

Cupertino, California

## INTRODUCTION

Numerical methods are generally judged on their ability to produce a reasonably accurate answer in a reasonable amount of time. This paper deals with the latter criterion in the context of non-standard architectures—namely vector and array processors. These architectures owe their very existence to the demands for more computing power than is available on conventional, sequential computers, so speed and efficiency are especially relevant in this area.

The major characteristics of array and vector processors are described in the next section. We then indicate how these features affect the formulation of efficient parallel algorithms. This leads to considerations involving information flow during a parallel algorithm and its effects on bandwidth requirements. The concluding section suggests the relevance of this work to the explosive growth in integrated circuit technology.

## ARRAY AND VECTOR PROCESSORS

Two types of parallel computer architectures are considered in this paper: the array processor and the vector, or pipelined, processor. Both have a centralized control facility so that in both architectures, only one instruction is being executed at a time, at least as far as the programmer is concerned. The machines get their parallel nature from the fact that one instruction may cause many pairs of operands to be combined to produce many results. The combining of the operands is logically simultaneous—all results seem to the programmer to be produced at once, although in reality their production may occur over a number of time periods with some being produced before others. It is the commonality of the two implementation approaches at the instruction level that makes it meaningful to talk about numerical techniques in common for both types of parallel computers.

Many of the algorithmic concerns of one type of processor are common to the other, although they may arise from different implementation details. This section presents a general description of array and vector processors, together with some general comments about how the implementations affect the design of numerical algorithms.

An array processor consists of a group of processing modules all under the control of a centralized control unit. The

central unit synchronizes the computation by issuing commands to the modules which in turn execute the commands on data in their respective memories. Thus when an array operation (say an add) is in progress,  $n$  pairs of operands will be processed, where  $n$  is the number of modules in the array. There is also the capability of selecting only a subset of the modules to be active for any command (either by the central unit or based upon local data in each module). When data must be transferred among modules, an interconnection network is used. The network is also under the control of the central control unit—the central unit commands the modules to place data to be transferred into a network output register, then causes all data to be transferred simultaneously and finally commands the modules to read the new data from their network input registers. An example of an array processor is the ILLIAC IV<sup>1</sup> which is an array of sixty-four modules having a two-dimensional interconnection network.

A vector processor consists of a central computing unit and a main memory. Data resides in the memory and is brought into the central computing unit in streams to be processed. While the central computing unit is processing some operands, computed results will be streaming back to the memory and additional operands will be streaming to the computing unit for processing. Typically, eight to thirty-two different pairs of operands will be in some stage of processing in the central computing unit during any one time during the execution of a vector instruction. As with the array processor, there is a method of preventing a subset of the result vector from being modified by computed results; in the vector processor this is done by means of a control vector. A vector of operands is usually restricted to be data in contiguous memory locations or in memory locations separated by a constant distance. Examples of vector processors are the Cyber 203 (see STAR-100)<sup>2</sup> and the CRAY-1.<sup>3</sup>

An array architecture is predicated on the advantages of replicating identical devices: economies of scale and simplicity of control. Economies of scale pertain both to manufacture and to maintenance: replication of identical modules as opposed to different types of modules implies fewer different types of modules to design, fewer different types of modules to understand how to repair, and the implied increase in volume suggests faster rise on the learning curve to manufacture. Simplicity of control stems from a control unit that treats all array modules the same; thus controlling an array with twice the number of modules (twice the max-

imum computing power) is logically the same as controlling a smaller configuration. Indeed, conceptually the processing power of an array processor can be increased simply by adding more modules until money runs out or, more likely, until the mean time between failures becomes intolerable.

A vector architecture is predicated on the advantages of centralized processing—flexibility of use and sophistication of available operations. Flexibility arises from there being no “natural” vector length as in an array processor where the number of modules defines the basic extent of parallelism. Vector processors can easily change the length of the vectors they process during the course of the computation, as required by the algorithm. And by centralizing the computing power, complex operations become possible—once data has been brought to the centralized arithmetic unit, it can be used in complex operations, perhaps iteratively, before being returned to the main memory. In contrast, iterative use of computed data in a sequential fashion seriously degrades the performance of array architectures.

#### IMPLICATIONS FOR ALGORITHM DESIGN

There are certain characteristics that both types of processors have in common from the point of view of algorithm design. The most important is that their parallel computing rate is much greater than their sequential or scalar computing rate—this is, in fact, their primary reason for existence. In current machines, the difference in computing power for parallel vs. scalar ranges from a factor of five (for the CRAY-1) to a factor of a hundred (for the ILLIAC IV) with a factor of twenty being a convenient rule of thumb.

For the algorithm designer, this fact translates into a goal of trading scalar formulations for vector formulations until a point of diminishing returns is reached. A simple example will clarify this principle. Suppose a function is to be evaluated at each point of a grid, with the values dependent only on the value of a parameter at that grid point. Thus all functions may be evaluated in parallel. In a sequential computing environment, a typical approach would be to divide the range of the parameter into subsets (sub-ranges) and find an efficient numerical approximation for each subset. Assume that a continued fraction approximation is chosen, with each sub-range determining how many terms to use in the evaluation. This may lead to a function where 50 percent of the calls require 20 time units, 30 percent require 40, 10 percent require 80, 5 percent require 160, 4 percent require 320 and 1 percent require 640. Thus evaluating the function at 100 points will take an average of 5720 time units.

A straightforward translation of this approach to a vector algorithm would be to use the worst case for all points. Since this will be done in vector mode, the time required is only 32 time units per point, or a total time of 3200 time units for 100 points (using the ratio factor of 20 mentioned above). This is about a 44 percent decrease in execution time. A better re-formulation would be to use the next to worst case assumption (the form that works for 99 percent of the points) for the entire grid, and then go back and correct the 1 percent that require a more accurate approximation. This reduces

the average time to 2240 time units, neglecting the overhead to find that 1 percent.

An even better solution might be to try to find a single approximation sufficiently accurate over the entire range—for example one that requires 120 time units per evaluation in sequential mode. For sequential computing, this would be a poor choice, but in the parallel processing context it takes only 600 time units to evaluate the function at 100 grid points, a definite improvement over the first adaptation of the algorithm.

In both array and vector architectures, the key to a good parallel algorithm is the partitioning of the task into a large set of identical operations on different, independent data sets. In an array architecture this leads to putting each data set in a separate module and then executing the same program in each module. In a vector processor this leads to putting each data set in a separate index of a set of vectors and then executing a program using vector instructions rather than scalar instructions.

In an array processor, once the number of identified independent tasks equals the number of modules there is little incentive in further refinements. However, in a vector processor, there is the phenomenon of vector start-up which encourages the use of long vectors (many independent tasks). With each vector instruction, there is an initial period of time while the vector processor is establishing data paths and computing the first element of the result vector. This time is generally lost, so that if one can combine several vector instructions into one, then the number of lost start-ups can be reduced to one.

Thus, for example, if one is to perform a succession of discrete Fourier transforms on different sets of data, it is beneficial to process them together, since the vectors will be  $M$  times as long, where  $M$  is the number of data sets. In practice, it turns out that when  $M$  is larger than five on the Cyber 203, executing the standard fast Fourier transform on  $M$  sets simultaneously is faster than evaluating a specially designed parallel discrete Fourier transform on the data sets (developed by Pease with an eye to using long vectors).<sup>4</sup>

As another example of the use of long vectors, consider the multiplication of two banded matrices. Matrix multiplication is traditionally expressed in terms of the inner product of rows of the first matrix and columns of the second matrix. But in the case of matrices of low bandwidth, this approach leads to short vectors and is therefore ill suited to parallel computers. Instead, consider the banded matrix as a sum of diagonal matrices. Matrix multiplication in this case takes the form of multiplication of diagonal matrices which is equivalent to component-wise multiplication of vectors. If the size of the matrix is much larger than its bandwidth, this leads to a substantial improvement in the time required to multiply banded matrices. This example will be used again in the discussion of information bandwidth in the next section.

#### INFORMATION FLOW IN PARALLEL ALGORITHMS

In a parallel processing environment, one is acutely aware of the placement and movement of data during a computa-

tion. In an array processor, data to be combined that reside in different modules must be sent to a single module to be processed. In a vector processor, vectors can be fetched most efficiently if they reside in contiguous memory locations. In both architectures, these considerations often lead to the rearrangement of data between steps in an algorithm.

The traditional example of data rearrangement is a matrix transpose. In an array processor, assume that each column of an  $n$  by  $n$  matrix resides in a separate module and that the matrix must be stored by row for the next step in an algorithm. Transposition of a matrix can be viewed as interchanging its diagonals. Each step of the process will choose a super-diagonal and a sub-diagonal to interchange. As the diagonals away from the main diagonal are progressively shorter, it is in fact possible to interchange two pairs of diagonals simultaneously, thereby making maximum use of the processor's bandwidth.

In a vector processor, there is usually a vector instruction that will gather up elements from random memory locations (or store a vector into arbitrary locations), as indicated by a target vector. Because of the unstructured or non-local distribution of operands, such instructions generally take much longer than standard vector instructions (a factor of ten to twenty on the Cyber 203, for example). One could transpose a matrix in this fashion, first generating the target addresses and then effecting the chaotic move. However, for matrices with dimensions a power of two, another method is attractive. If the indices of the matrix are considered as bit strings, then transposing the matrix corresponds to interchanging these bit patterns, or a series of rotations of these bit patterns. Such data re-arrangement corresponds to successive applications of a perfect shuffle data re-arrangement.<sup>5</sup> Thus matrix transposition can be performed in  $\log_2 n$  steps, where  $n$  is the smaller dimension, each step operating on vectors of length  $nm$ , where  $m$  is the other dimension. Note that this algorithm deals with considerably longer vectors than an adaptation of the matrix transpose operation given for array architectures. The major disadvantage of this second algorithm is that it requires several passes through the entire matrix; too many such passes (large values of  $n$ ) make the first, random gather algorithm faster.

The movement of information during an algorithm deserves careful scrutiny in a parallel algorithm. In an array processor, this is essentially the movement of data among the modules. In a vector processor, this is essentially the rearrangement of data to form contiguous vectors.

For example, consider a two dimensional fast Fourier transform, which is the tensor product of one dimensional fast Fourier transforms. With regard to data movement, the two dimensional fast Fourier transform on an  $n$  by  $n$  data set is equivalent to  $n$  simultaneous one dimensional transforms along one dimension, followed by  $n$  simultaneous one dimensional transforms along the other dimension. If the data set is arranged so that one dimension is across array modules and the other is within modules, then the first step involves executing the transform in parallel on data contained entirely within modules—no transfer of data among modules is required. The next step requires  $n$  more transforms, and these may be done successively, using a parallel

transform where each transform takes one datum from each module. This entails on the order of  $\log_2 n$  transfers of data among modules per transform. An alternative is to transpose the data and repeat the first step. Jesshope<sup>6</sup> has shown that if the array is interconnected as a  $k$ -dimensional grid (for any  $k$ ) then the better method is to transpose and repeat the transform within modules.

Jesshope's work assumes that the data are stored in a "natural" order, that is, the processors are numbered such that adjacent processors have numbers that differ by a constant and the constant is the same for each processor; also, the  $(i, j)$  data element is assigned to processor  $i$ . The question arises as to lifting this last restriction; that is, to allow the data to be assigned to processors so as to minimize the time required for data inter-communication. Although for some types of data manipulation this idea can produce substantial time savings, the work by Kung and Stevenson<sup>7</sup> suggests that this is not the case for the fast Fourier transform.

As another example of information flow during the course of a parallel algorithm, and a measure of the bandwidth required by the two architectures to support the execution of a parallel formulation of the problem, consider the above example of multiplying two banded matrices. For the following discussion, the equations of matrix multiplication for tridiagonal matrices  $A$  and  $B$  will be used. The method is expressed in vector notation where  $a_d$  is the main diagonal of  $A$ ,  $a_{d-1}$  is the first sub-diagonal and  $a_{d+1}$  is the first super-diagonal. Thus  $a_d(i)$  corresponds to  $a(i, i)$  and  $a_{d-1}(i)$  corresponds to  $a(i, i-1)$ . The equations are

$$\begin{aligned} c_{d+2}(i) &= a_{d+1}(i) b_{d+1}(i+1) \\ c_{d+1}(i) &= a_d(i) b_{d+1}(i) + a_{d+1}(i) b_d(i+1) \\ c_d(i) &= a_{d-1}(i) b_{d+1}(i-1) + a_d(i) b_d(i) \\ &\quad + a_{d+1}(i) b_{d-1}(i+1) \\ c_{d-1}(i) &= a_{d-1}(i) b_d(i-1) + a_d(i) b_{d-1}(i) \\ c_{d-2}(i) &= a_{d-1}(i) b_{d-1}(i-1) \end{aligned} \quad (1)$$

In this form the algorithm can be implemented on either a vector or an array computer. On a vector processor, nine vector multiplies and four vector adds are indicated. Current vector processors, such as the Cyber 203, permit only one vector operation per pass through the central computing unit (on the CRAY-1, a multiply and an add may be performed under certain conditions). Thus the number of operands streaming to and from memory during the course of this algorithm is approximately  $39n$  where  $n$  is the length of the main diagonal of the matrices. Because of the centralization of the computing, this means that the bandwidth of the central unit must be tremendous for high performance, and in fact becomes the bottleneck in constructing very large systems based on a vector architecture.

Now consider the same algorithm on an array processor. Here the  $i^{\text{th}}$  element of each vector is stored in module number  $i$ . For the computation, six numbers must be exchanged among modules, and this can be done in six parallel steps. Thus the number of operands that must flow during the

course of the algorithm is approximately  $6n$ . This is considerably less than the case for a vector processor, and the required bandwidth of the array is distributed among the modules rather than concentrated at a centralized point. This means that the bandwidth between any two modules can be rather pedestrian and still maintain high performance of the machine.

The natural question which arises at this point is whether there is a way to reduce the bandwidth requirement between the main memory and the centralized computing unit in a vector processor. The answer is yes: do more with each datum before returning the intermediate results to memory. This is the approach taken in the BSP,<sup>8</sup> where up to five streams of operands may be combined by one vector instruction to produce one result (e.g., one BSP instruction forms  $A + B * C$  for vectors  $A$ ,  $B$  and  $C$ , a primitive operation that is useful in matrix multiplication, for example).

The major difficulty with his approach lies in the complexity of the centralized computing unit, which must now have internal registers to hold temporary results and numerous data and control paths to implement many templates for combining operands.

Work by Kung and Leiserson<sup>9</sup> shows that a regular array of simple elements can be used for the centralized computing unit to perform several useful and powerful operations: matrix multiplication, LU decomposition, and finite impulse response calculations are a few of their examples. All these algorithms require only one pass through the data.

Each of the simple elements in their array is designed to perform a simple basic operation  $A + B * C$  where  $A$ ,  $B$  and  $C$  are inputs. Outputs of the elements are  $B$ ,  $C$  and the computed value. Other capabilities are needed for some of the more complex examples, but this capability is enough to multiply two tri-diagonal matrices. The diagonal elements of matrices  $A$  and  $B$  stream into the unit and the product  $C$  is computed, again by diagonals. Equations (1) are used to compute the products.

The important point of this example is that the data are fetched from memory exactly once, reducing the bandwidth demands on the memory; the bandwidth available within the array is actually greater than needed, but is distributed among the modules of the array and therefore is relatively low. The second key point is that the centralized computing unit is a regular array, but a larger configuration must be used to process larger matrices at maximum efficiency. Thus this approach achieves the advantage of the vector processor in handling variable sized vectors and the advantage of the array processor with its regular structure, replicated units and lower inter-module bandwidth requirements.

One drawback which still remains, however, is that processing time is still a function of the vector length since entire vectors must be pushed through the centralized unit. For some problems, this is unnecessary, as the next example shows.

The structures of Kung and Leiserson also lend themselves to solving certain problems with an inherently recursive formulation, provided their modules are given additional capabilities of operations and mapping of results to

outputs (the latter is needed to effect the loops implied by the recursive use of computed results). One example of recursion is the LU decomposition of a matrix, which illustrates the last topic in this paper.

The LU decomposition of a tri-diagonal matrix  $C$  is specified as a lower bi-diagonal matrix  $A$  with ones on the diagonal and an upper bi-diagonal matrix  $B$  such that  $C = A * B$ , where  $*$  is now matrix multiplication. The equations defining  $A$  and  $B$  can be expressed as

$$\begin{aligned} a_{d-1}(i) &= c_{d-1}(i)/b_d(i-1) \\ b_d(i) &= c_d(i) - a_{d-1}(i)c_{d+1}(i-1) \\ b_{d+1}(i) &= c_{d+1}(i) \end{aligned} \quad (2)$$

Note that the definition of  $b_d(i)$  depends through  $a_{d-1}(i)$  on the value of  $b_d(i-1)$ . In the Kung and Leiserson array, this computed quantity can be fed back into the calculation as well as returned to memory, indicating the adaptability of their approach.

However, if equations (2) are considered as specifying the computation being performed by module number  $i$  in an array processor, this recursive dependency means that module  $i$  will require the correct input from module  $i-1$  before it can produce correct values for its components of  $A$  and  $B$ . Thus it will take time proportional to the number of modules (length of the vectors) for this information to trickle from the first module to the last.

While waiting for the correct value of  $b$ , the modules are idle, with no useful work. What happens if they perform the same operation as if they had the correct data—that is, fetch a value  $b_d$  from the predecessor module, compute  $a_{d-1}$  and  $b_d$  and pass their newly computed value of  $b_d$  to the successor module?

As shown by Heller, Stevenson and Traub,<sup>10</sup> if the matrix  $C$  is diagonally dominant, successive computed values of  $A$  and  $B$  are closer to the solution. And depending on the amount of diagonal dominance, the convergence can be rapid enough to terminate the process well before  $n$  steps (where  $n$  is the size of the system), depending upon the degree of accuracy needed. The interesting point of this work is an indication of how far information must travel in computing a suitably accurate answer, which need not be the distance required for an exact answer using exact arithmetic.

## CONCLUSION

The advent of sophisticated microprocessors and specialized chips capable of floating point arithmetic signal a maturing technology well suited for large arrays dedicated to parallel processing. Experience with the large scale parallel processors has indicated several general techniques or guidelines for developing parallel algorithms for such configurations, as discussed in this paper, but more work needs to be done if these new structures are to be used effectively in many applications. New numerical algorithms are needed that are better suited for these parallel architectures than are conventional sequential approaches.

## REFERENCES

1. Barnes, G. H., Brown, R. M., Kato, M., Kuck, D. J., Slotnick D. L. and Stokes, R. A., "The ILLIAC IV Computer," *IEEE Transactions on Computers*, Vol. C-17, 1968, p. 746.
2. Holland, S. A. and Purcell, C. J., "The CDC STAR-100: A Large Scale Network Oriented Computer System," *Proceedings IEEE Conference (Boston)*, 1971, p. 55.
3. Russell, R. M., "The CRAY-1 Computer System," *Communications of ACM*, Vol. 21, 1978, p. 423.
4. Korn, D. G. and Lambiotte, J. J., "Computing the Fast Fourier Transform on a Vector Computer," *Mathematics of Computing*, Vol. 33, 1979, p. 977.
5. Stone, H. S., "Parallel Processing with the Perfect Shuffle," *IEEE Transactions on Computers*, Vol. C-20, 1971, p. 153.
6. Jesshope, C. R., "Implementation of Fast Radix 2 Transforms on Array Processors," Department of Computer Science Technical Report, University of Reading, 1977.
7. Kung, H. T. and Stevenson, D. K., "A Software Technique for Reducing the Routing Time on a Parallel Computer with a Fixed Interconnection Network," *Proceedings of a Symposium on High Speed Computer and Algorithm Organization*, Academic Press, New York, 1978, p. 423.
8. Stokes, R. A., "Burroughs Scientific Processor," *Proceedings of a Symposium on High Speed Computer and Algorithm Organization*, Academic Press, New York, 1978, p. 85.
9. Kung, H. T. and Leiserson, C. E., "Algorithms for VLSI Processor Arrays," *Introduction to VLSI Systems*, Addison Wesley, Reading MA, 1980, p. 271.
10. Heller, D. E., Stevenson, D. K. and Traub, J. F., "Accelerated Iterative Methods for the Solution of Tridiagonal Systems on Parallel Computers," *Journal of ACM*, Vol. 23, 1976, p. 636.



# Design of an interactive matrix calculator

by CLEVE MOLER

University of New Mexico  
Albuquerque, New Mexico

## INTRODUCTION

MATLAB is an interactive computer program that serves as a convenient "laboratory" for computations involving matrices. It provides easy access to matrix software developed by the LINPACK and EISPACK projects [1-3]. The capabilities range from standard tasks such as solving simultaneous linear equations and inverting matrices, through symmetric and nonsymmetric eigenvalue problems, to fairly sophisticated matrix tools such as the singular value decomposition.

It is expected that one of MATLAB's primary uses will be in the classroom. It should be useful in introductory courses in applied linear algebra, as well as more advanced courses in numerical analysis, matrix theory, statistics and applications of matrices to other disciplines. In nonacademic settings, MATLAB can serve as a "desk calculator" for the quick solution of small problems involving matrices.

The program is written in Fortran and is designed to be readily installed under any operating system which permits interactive execution of Fortran programs. The resources required are fairly modest. There are about 6000 lines of Fortran source code, including the LINPACK and EISPACK subroutines used. With proper use of overlays, it is possible to run the system on a minicomputer with only 32K bytes of memory.

The size of the matrices that can be handled in MATLAB depends upon the amount of storage that is set aside when the system is compiled on a particular machine. We have found that an allocation of 4000 words for matrix elements is usually quite satisfactory. This provides room for several 20 by 20 matrices, for example. One implementation on a virtual memory system provides 50,000 elements. Since most of the algorithms used access memory in a sequential fashion, the large amount of allocated storage causes no difficulties.

In some ways, MATLAB resembles SPEAKEASY [4] and, to a lesser extent, APL. All are interactive terminal languages that ordinarily accept single-line commands or statements, process them immediately, and print the results. All have arrays or matrices as principal data types. But for MATLAB, the matrix is the only data type (although scalars, vectors and text are special cases), the underlying system is portable and requires fewer resources, and the supporting

subroutines are more powerful and, in some cases, have better numerical properties.

Together, LINPACK and EISPACK represent the state of the art in software for matrix computation. EISPACK is a package of over 70 Fortran subroutines for various matrix eigenvalue computations that are based for the most part on Algol procedures published by Wilkinson, Reinsch and their colleagues [5]. LINPACK is a package of 40 Fortran subroutines (in each of four data types) for solving and analyzing simultaneous linear equations and related matrix problems. Since MATLAB is not primarily concerned with either execution time efficiency or storage savings, it ignores most of the special matrix properties that LINPACK and EISPACK subroutines use to advantage. Consequently, only 8 subroutines from LINPACK and 5 from EISPACK are actually involved.

This paper gives a brief description of MATLAB from the user's point of view and presents a formal description of the MATLAB language. The system was designed and programmed using techniques described by Wirth [6], implemented in nonrecursive, portable Fortran.

## 1. Elementary operations

MATLAB works with essentially only one kind of object, a rectangular matrix with complex elements. If the imaginary parts of the elements are all zero, they are not printed, but they still occupy storage. In some situations, special meaning is attached to 1 by 1 matrices, that is scalars, and to 1 by  $n$  and  $m$  by 1 matrices, that is row and column vectors.

Matrices can be introduced into MATLAB in four different ways:

- Explicit list of elements,
- Use of FOR and WHILE statements,
- Read from an external file,
- Execute an external Fortran program.

The explicit list is surrounded by angle brackets, ' $\langle$ ' and ' $\rangle$ ,' and uses the semicolon ';' to indicate the ends of the rows. For example, the input line

$$A = \langle 1 \ 2 \ 3; \ 4 \ 5 \ 6; \ 7 \ 8 \ 9 \rangle$$



will result in the output

```
A = 1.  2.  3.
      4.  5.  6.
      7.  8.  9.
```

The matrix  $A$  will be saved for later use. The individual elements are separated by commas or blanks and can be any MATLAB expressions, for example

```
x = (-1.3, 4/5, 4*atan(1))
```

results in

```
X = -1.3000  0.8000  3.1416
```

The elementary functions available include sqrt, log, exp, sin, cos, atan, abs, round, real, imag, and conjg.

The FOR statement allows the generation of matrices whose elements are given by simple formulas. The above matrix  $A$  could also have been produced by

```
for i = 1:3, for j = 1:3, A(i,j) = 3*(i-1) + j;
```

The semicolon at the end of the line suppresses the printing, which in this case would have been nine versions of  $A$  with changing elements. Several statements may be given on a line, separated by semicolons or commas.

Names of variables are formed by a letter, followed by any number of letters and digits, but only the first four characters are remembered.

The special character prime (') is used to denote the transpose of a matrix, so

```
x = x'
```

changes the row vector above into the column vector

```
X = -1.3000
      0.8000
      3.1416
```

Addition, subtraction and multiplication of matrices are denoted by +, -, and \*. The operations are performed whenever the matrices have the proper dimensions. For example, with the above  $A$  and  $x$ , the expressions  $A + x$  and  $x * A$  are incorrect because  $A$  is 3 by 3 and  $x$  is now 3 by 1. However,

```
b = A*x
```

is correct and results in the output

```
B = 9.7248
      17.6496
      25.5743
```

Note that both upper and lower case letters are allowed for input (on those systems which have both), but that lower case is converted to upper case.

There are two "matrix division" symbols in MATLAB, \ and /. If  $A$  and  $B$  are matrices, then  $A \setminus B$  and  $B / A$  correspond formally to left and right multiplication of  $B$  by the inverse of  $A$ , that is  $\text{inv}(A) * B$  and  $B * \text{inv}(A)$ , but the result is obtained directly without the computation of the inverse. In the scalar

case,  $3/1$  and  $1/3$  have the same value, namely one-third. In general,  $A \setminus B$  denotes the solution  $X$  to the equation  $A * X = B$  and  $B / A$  denotes the solution to  $X * A = B$ .

Left division,  $A \setminus B$ , is defined whenever  $B$  has as many rows as  $A$ . If  $A$  is square, it is factored using Gaussian elimination. The factors are used to solve the equations  $A * X(:, j) = B(:, j)$  where  $B(:, j)$  denotes the  $j$ th column of  $B$ . The result is a matrix  $X$  with the same dimensions as  $B$ . If  $A$  is nearly singular (according to the LINPACK condition estimator, RCOND), a warning message is printed. If  $A$  is not square, it is factored using Householder orthogonalization with column pivoting. The factors are used to solve the overdetermined equations in a least squares sense. The result is an  $m$  by  $n$  matrix  $X$  where  $m$  is the number of columns of  $A$  and  $n$  is the number of columns of  $B$ . Each column of  $X$  has at most  $k$  nonzero components, where  $k$  is the effective rank of  $A$ .

Right division,  $B / A$ , can be defined in terms of left division by  $B / A = (A \setminus B)'$ .

The expression  $A ** p$  means  $A$  to the  $p$ th power. It is defined if  $A$  is a square matrix and  $p$  is a scalar. If  $p$  is an integer greater than one, the power is computed by repeated multiplication. For other values of  $p$  the calculation involves the eigenvalues and eigenvectors of  $A$ .

There are three predefined variables, RAND, EYE and FLOP. The value of RAND is a random variable, with a choice of a uniform or a normal distribution. The name EYE is used in place of  $I$  to denote identity matrices because  $I$  is often used as a subscript or as  $\text{sqrt}(-1)$ . The dimensions of EYE are determined by context. For example,

```
B = A + 3 * EYE
```

adds 3 to the diagonal elements of  $A$  and

```
X = EYE / A
```

is one of several ways in MATLAB to invert a matrix.

FLOP provides a count of the number of floating point operations, or "flops," required for each calculation.

All computations are done using either single or double precision real arithmetic, whichever is appropriate for the particular computer. There is no mixed-precision arithmetic. The Fortran COMPLEX data type is not used because many systems create unnecessary underflows and overflows with complex operations and because some systems do not allow double precision complex arithmetic.

## 2. MATLAB functions

Much of MATLAB's computational power comes from the various matrix functions available. The current list includes:

- INV(A)—Inverse.
- DET(A)—Determinant.
- COND(A)—Condition number.
- RCOND(A)—A measure of nearness to singularity.
- EIG(A)—Eigenvalues and eigenvectors.
- SCHUR(A)—Schur triangular form.
- POLY(A)—Characteristic polynomial.

SVD(A)—Singular value decomposition.  
 PINV(A,eps)—Pseudoinverse with optional tolerance.  
 RANK(A,eps)—Matrix rank with optional tolerance.  
 LU(A)—Factors from Gaussian elimination.  
 CHOL(A)—Factor from Cholesky factorization.  
 QR(A)—Factors from Householder orthogonalization.  
 RREF(A)—Reduced row echelon form.  
 ORTH(A)—Orthogonal vectors spanning range of A.  
 EXP(A)— $e$  to the A.  
 LOG(A)—Natural logarithm.  
 SQRT(A)—Square root.  
 SIN(A)—Trigonometric sine.  
 COS(A)—Cosine.  
 ATAN(A)—Arctangent.  
 ROUND(A)—Round the elements to nearest integers.  
 ABS(A)—Absolute value of the elements.  
 REAL(A)—Real parts of the elements.  
 IMAG(A)—Imaginary parts of the elements.  
 CONJG(A)—Complex conjugate.  
 SUM(A)—Sum of the elements.  
 PROD(A)—Product of the elements.  
 DIAG(A)—Extract or create diagonal matrices.  
 NORM(A,p)—Norm with  $p=1, 2$  or 'Infinity.'  
 EYE(m,n)—Portion of identity matrix.  
 RAND(m,n)—Matrix with random elements.  
 ONES(m,n)—Matrix of all ones.  
 MAGIC(n)—Interesting test matrices.  
 HILBERT(n)—Inverse Hilbert matrices.  
 ROOTS(C)—Roots of polynomial with coefficients C.  
 USER(A)—Function defined by external Fortran program.

Some of these functions have different interpretations when the argument is a matrix or a vector and some of them have additional optional arguments.

Several of these functions can be used in a generalized assignment statement with two or three variables on the left hand side. For example

$$\langle X, D \rangle = \text{EIG}(A)$$

stores the eigenvectors of A in the matrix X and a diagonal matrix containing the eigenvalues in the matrix D. The statement

$$\text{EIG}(A)$$

simply computes the eigenvalues and stores them in ANS.

### 3. Rows, columns and submatrices

Individual elements of a matrix can be accessed by giving their subscripts in parentheses, e.g.,  $A(1,2)$ ,  $x(i)$ ,  $\text{TAB}(\text{index}(k-1)+1)$ . An expression used as a subscript is rounded to the nearest integer.

Individual rows and columns can be accessed using a colon ':' for the free subscript. For example,  $A(1,:)$  is the first row of A and  $A(:,j)$  is the  $j$ th column. Thus

$$A(i,:) = A(i,:) + c * A(k,:)$$

adds  $c$  times the  $k$ th row of A to the  $i$ th row.

The colon is used in several other ways in MATLAB, but all of the uses are based on the following definition.

$j:k$  is the same as  $\langle j, j+1, \dots, k \rangle$

$j:k$  is empty if  $j > k$ .

$j:i:k$  is the same as  $\langle j, j+i, j+2i, \dots, k \rangle$

$j:i:k$  is empty if  $i > 0$  and  $j > k$  or if  $i < 0$  and  $j < k$ .

The colon is usually used with integers, but it is possible to use arbitrary real scalars as well. Thus

$1:4$  is the same as  $\langle 1, 2, 3, 4 \rangle$

$0:0.1:0.5$  is the same as  $\langle 0.0, 0.1, 0.2, 0.3, 0.4, 0.5 \rangle$

In general, a subscript can be a vector. If X and V are vectors, then  $X(V)$  is  $\langle X(V(1)), X(V(2)), \dots, X(V(n)) \rangle$ . This can also be used with matrices. If V has  $m$  components and W has  $n$  components, then  $A(V,W)$  is the  $m$  by  $n$  matrix formed from the elements of A whose subscripts are the elements of V and W. Combinations of the colon notation and the indirect subscripting allow manipulation of various submatrices. For example,

$A(\langle 1, 5 \rangle, :)$  =  $A(\langle 5, 1 \rangle, :)$  interchanges rows 1 and 5 of A.

$A(2:k, 1:n)$  is the submatrix formed from rows 2 through  $k$  and columns 1 through  $n$  of A.

$A(:, \langle 3 \ 1 \ 2 \rangle)$  is a permutation of the first three columns of A.

### 4. FOR, WHILE and IF

The FOR clause allows statements to be repeated a specific number of times. The general form is

FOR variable = expr, statement, ..., statement, END

The END and the comma before it may be omitted. In general, the expression may be a matrix, in which case the columns are stored one at a time in the variable and the following statements, up to the END or the end of the line, are executed. The expression is often of the form  $j:k$ , and its "columns" are simply the scalars from  $j$  to  $k$ . Some examples (assume  $n$  has already been assigned a value):

$A = \text{EYE}(n)$ ; for  $i = 1:n$ , for  $j = 1:n$ ,  $A(i, j) = 1/(i+j-1)$ ;

generates the Hilbert matrix.

for  $j = 2:n-1$ , for  $i = j:n-1$ ,  $A(i, j) = 0$ ; end;  $A(j, j) = j$ ;

end; A

changes all but the "outer edge" of the lower triangle and then prints the final matrix.

for  $h = 1:0: -0.1: -1.0$ ,  $((h, \cos(\text{pi}*h)))$

prints a table of cosines.

$\langle X, D \rangle = \text{EIG}(A)$ ; for  $v = X, v, A * v$

displays eigenvectors, one at a time.

The WHILE clause allows statements to be repeated an indefinite number of times. The general form is

```
WHILE expr relop expr, statement, ..., statement,
  END
```

where relop is =, <, >, <=, >=, or <> (not equal). The statements are repeatedly executed as long as the indicated comparison between the real parts of the first components of the two expressions is true. Here are two examples. (Exercise for the reader: What do these segments do?)

```
eps = 1;
while 1 + eps > 1, eps = eps/2;
eps = 2*eps
```

```
E = 0*A; F = E + EYE; n = 1;
while NORM(E + F - E, 1) > 0, E = E + F; F = A*F/n;
  n = n + 1;
E
```

The IF clause allows conditional execution of statements. The general form is

```
If expr relop expr, statement, ..., statement, ELSE
  statement, ..., statement
```

The first group of statements is executed if the relation is true and the second group is executed if the relation is false. The ELSE and the statements following it may be omitted. For example,

```
if abs(i-j) = 2, A(i,j) = 0;
```

### 5. Commands, text and files

MATLAB has several commands which control the output format and the overall execution of the system.

The HELP command allows on-line access to short portions of text describing various operations, functions and special characters.

Results are usually printed in a scaled fixed point format that shows four or five significant figures. The commands SHORT, LONG, SHORT E, and LONG E alter the output format, but do not alter the precision of the computations or the internal storage.

The command CHOP(*p*) causes *p* octal or hexadecimal figures to be chopped off after each subsequent floating point operation, thereby simulating a computer with a shorter word length. CHOP(0) restores full accuracy.

The CLEAR command erases all stored variables, except FLOP, RAND and EYE. The statement  $A = \langle \rangle$  indicates that a "0 by 0" matrix is to be stored in A. This causes A to be erased so that its storage can be used for other variables.

MATLAB has a limited facility for handling text. Any string of characters delimited by quotes (with two quotes used to allow one quote within the string) is saved as a vector of integer values. For example

'A = 2 + 2' is the same as (10 36 46 36 2 36 41 36 2)

It is possible, though seldom very meaningful, to use such strings in matrix operations. More frequently, the text is used as a special argument to various functions.

NORM(A, 'inf') computes the infinity norm of A.

EXEC(T) replaces the remainder of the input line with the text stored in T.

EXEC('file') obtains subsequent MATLAB input from an external file.

SAVE('file') stores all the current variables, pointers, etc. in an external file.

LOAD('file') retrieves everything stored by a previous SAVE('file')

PUT('file', X) writes X on a file so that it can be retrieved with GET or accessed by another program.

X = GET('file') reads X from a file where it was placed by PUT or another program.

The operations which access external files cannot be handled in a completely machine-independent manner by portable Fortran code. It is necessary for each particular installation to provide a subroutine which associates external text files with Fortran logical unit numbers.

### 6. Syntax diagrams

A formal description of the language acceptable to MATLAB, as well as a flow chart of the MATLAB program, is provided by the syntax diagrams or syntax graphs of Wirth [6]. There are ten nonterminal symbols in the language:

line, statement, clause, expression, term, factor, number, integer, name, command.

The following syntax diagrams define each of the nonterminal symbols using the others and the terminal symbols:

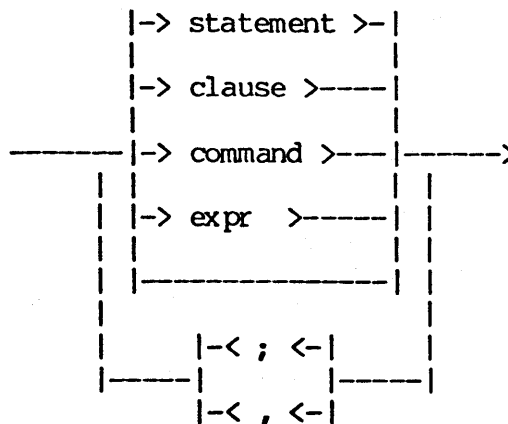
letter—A through Z,

digit—0 through 9,

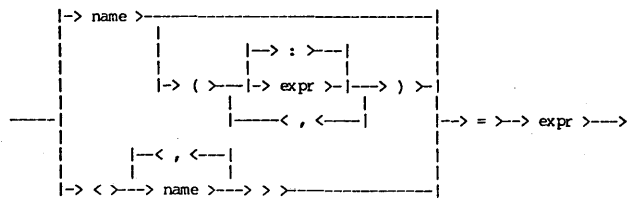
character—( ) ; : + - \* / \ = . , ' < >

text—any sequence of letters, digits, and characters.

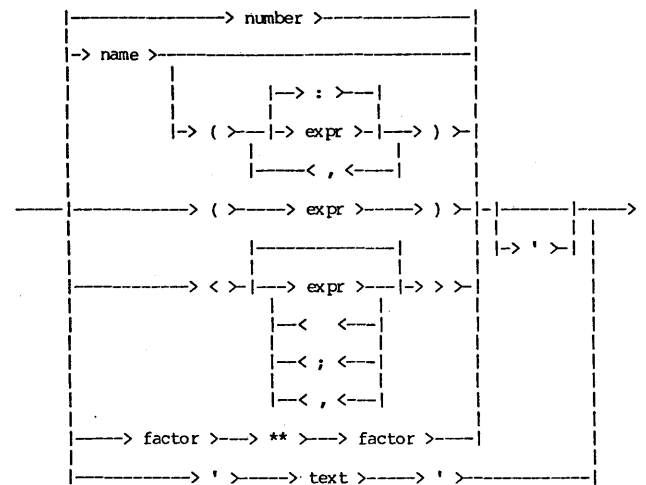
line



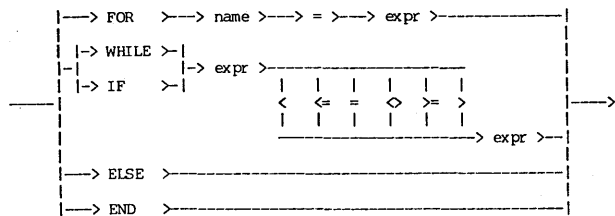
statement



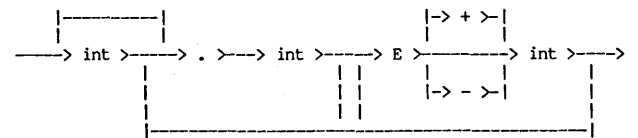
factor



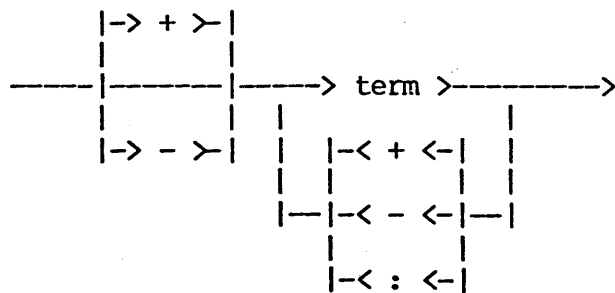
clause



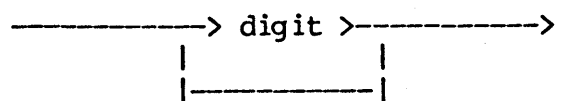
number



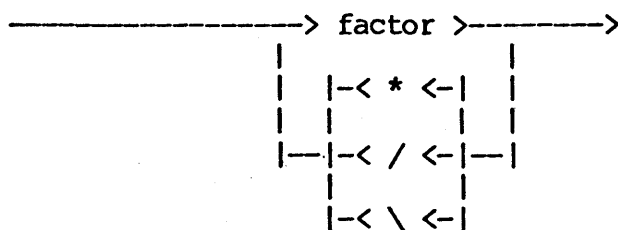
expr



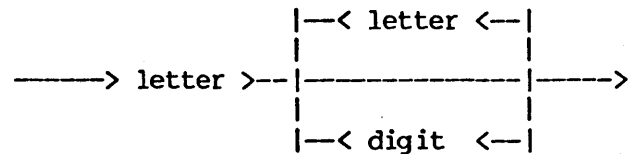
int



term



name



command



## ACKNOWLEDGMENT

Work on MATLAB has been carried out at the University of New Mexico, and during visits to Stanford Linear Accelerator Center, Argonne National Laboratory and Los Alamos Scientific Laboratory. Support has been provided by the National Science Foundation and the Department of Energy.

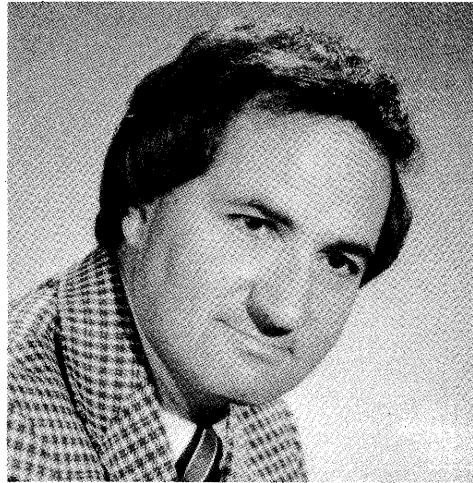
## REFERENCES

1. Dongarra, J. J., Bunch, J. R., Moler, C. B., and Stewart, G. W., "LINPACK Users' Guide," Society for Industrial and Applied Mathematics, Philadelphia, 1979.
2. Smith, B. T., Boyle, J. M., Dongarra, J. J., Garbow, B. S., Ikebe, Y., Klema, V. C., and Moler, C. B., "Matrix Eigensystem Routines—EISPACK Guide," *Lecture Notes in Computer Science*, volume 6, second edition, Springer-Verlag, 1976.
3. Garbow, B. S., Boyle, J. M., Dongarra, J. J., and Moler, C. B., "Matrix Eigensystem Routines—EISPACK Guide Extension," *Lecture Notes in Computer Science*, volume 51, Springer-Verlag, 1977.
4. Cohen, S. and Piper, S., "SPEAKEASY III Reference Manual," Speakeasy Computing Corp., Chicago, Ill., 1979.
5. Wilkinson, J. H. and Reinsch, C., "Handbook for Automatic Computation," volume II, *Linear Algebra*, Springer-Verlag, 1971.
6. Wirth, Niklaus, *Algorithms + Data Structures = Programs*, Prentice-Hall, 1976.

## **Image Processing**

NCC '80 has addressed three important areas of image processing through individually organized sessions. These topics are medical imaging, facsimile transmission, and image understanding. In addition, the panel discussion organized by T. Wiener addresses various current topics.

The addressed image processing activities have been strongly influenced by computer technology advances.



**Andrew Tescher**  
*Area Director*



# Derivation of invariant scene characteristics from images

by BERTHOLD K. P. HORN

*Artificial Intelligence Laboratory,  
Massachusetts Institute of Technology,  
Cambridge, Massachusetts*

## UNDERESTIMATING THE DIFFICULTIES

To us, vision is an immediate experience, not subject to careful introspection. We cannot write down protocols of processing steps that lie between the raw image intensities and our vivid impression of the surrounding scenery. Furthermore, we find ourselves in possession of this faculty long before we learn to master the more sequential processing tasks involved in the use of language, for example. Consequently, the difficulties of the vision process are often not appreciated. This is as true today, when many inroads have been made on the problem of understanding this process, as it was earlier when it was thought that vision could be understood simply in terms of some general ideas of artificial intelligence.

## DIVERGENCE OF OBJECTIVES

This difficulty is further compounded by the fragmentation of efforts resulting from widely varying motivations which bring researchers to this problem. These range from an intense desire to understand naturally occurring vision systems to an interest in industrial application of the machine vision. Somewhere in between we find those pursuing the information inherent in an image without regard to the implementation details of particular vision systems. It is not too surprising then that one finds widely diverging criteria for judging the importance of a particular piece of work.

### *Machine vision is not the "I/O of A.I."*

Still others use the vision domain only as a test bed to illustrate some general mechanisms currently favored in other artificial intelligence work, or think of machine vision and manipulation merely as the "I/O of A.I.," the interfaces which allow the smart machine to interact intelligently with its environment. I contend that this is unreasonable since vision appears to have interesting features which do not have counterparts elsewhere, certainly not in the serial, linguistic kind of reasoning pursued in other areas of artificial intelligence. It is these features which make vision worthy of study in its own right. One such aspect which has not been ap-

proached seriously elsewhere is that of spatial reasoning which cannot be conveniently handled using the kinds of data structures explored so far and found so useful in other domains.

### *Task of a vision system*

What is the task to be tackled by a vision system? Despite widespread disagreement on many other aspects of vision, most would agree that a vision system is expected to produce a description of what is being viewed. The input may be one or more images, each a two-dimensional distribution of scene radiance values obtained from some sensing device. There is less agreement on the form of the output. What kind of description is acceptable? Clearly two criteria must be satisfied: The description must

- (1) reflect some aspects of the three-dimensional reality, and
- (2) be useful in carrying out a specified task.

Usually it is expected that the description take a symbolic form. Quite different kinds of descriptions are likely to be considered adequate when the system is part of a device which lines up integrated circuit chips for automated lead bonding [Horn, 1975b], as opposed to a stage of a system meant to express an opinion about the merits of a work of art. As is so common, the representation of the given information must be matched to the task at hand.

### *Task independence*

It would be much nicer if one common mode of description could be employed, since the system dealing with visual inputs then could be designed in isolation, without considering the overall task. Perhaps this will turn out to be possible, at least for early stages of an image analysis system. Certain kinds of operations on images appear to be dictated by the image rather than the task and ought to be done without consideration for the task. At this point, however, it seems that task-dependent representations will be with us for a while.



## COMPLEXITY OF MACHINE VISION

An important lesson to be learned from the work on vision so far is that the problems are profound and not likely to succumb to the application of a bag of tricks from some other field, such as communications theory, statistics or linear systems theory. Machine vision merits its own methodology. Attention must be paid to both the physics of image formation as well as the information processing techniques which can produce the desired internal descriptions of what is being viewed. It now seems that this latter endeavor can be helped along by careful consideration of the human visual system and its strength and weaknesses.

Unfortunately biological vision systems are extremely complex and one can easily be led astray while studying them in isolation, without adequate tests of hypothesis one develops. Similarly, knowledge of certain physiological or chemical detail, for example, may not turn out to be very illuminating. What happens to the conformation of the rhodopsin molecule in the first few pico-seconds after a photon hits it is very exciting, but not helpful in the understanding of vision in a broader sense. One thing one learns quickly from even casual study of natural vision systems is that prodigious amounts of computation are involved in the processing of the image information.

### A PECULIAR DICHOTOMY

For a mobile biological entity above a certain size, vision is vital. It is hard to survive in a world where others have this faculty and use it in competition for food and in predator avoidance. Similarly, machine vision holds great promise for artificial systems. Many tasks cannot be done, or can be done only slowly or clumsily without it. So, vision, while difficult, is also very useful. As a result people will push the technology hard to get working systems. This has resulted in a peculiar dichotomy. There are two kinds of systems:

- (1) systems which work at reasonable speeds, and
- (2) systems which work reasonably well.

#### *"Automated" stereo*

Illustrations of this curious phenomena abound. There are, for example, a variety of machines which extract topographic information at reasonable speeds from stereo pairs of aerial photographs. These devices use special purpose hardware to implement rather simple correlation techniques, and, as a result, require significant human assistance. First, the operator is obliged to help the system out of "trouble spots" where the correlation technique fails because either there is no detail, as on smooth sand or a lake, or because the two views are too different, perhaps because the slope is large. Many times the machine does not even note that it is in trouble and so records bad information. These "glitches" then have to be tediously removed in an interactive editing process if the data is to be at all useful.

On the other side of the coin, one finds many good ideas in the machine vision community which require sophisticated hardware and software for their implementation and which are slow on computers of today's ilk. Methods recently developed at Stanford [Quam 1971, Gennery 1977, Arnold 1978] and at M.I.T. [Marr 1974, Marr & Poggio 1976, Marr & Poggio 1977] are considerably more robust, but require staggering computing power on machines of standard architecture.

#### *"Automatic" terrain classification*

Systems have been developed for classification of terrain based on the application of pattern recognition techniques on a point by point basis. Special hardware has even been built to implement this simple process, perhaps prematurely, since the performance of this method leaves much to be desired. The classifier has to be trained anew for each image; it cannot deal with hilly terrain and changes in the lighting angles. Typically, the classifier is only used as a step in an iterative refinement process with the human operator making the real decisions. Even then many points are incorrectly classified if the separation between classes is not very distinct.

Yet at the same time work at Purdue [Landgrebe 1973, Landgrebe 1975, Gupta et al. 1973, Swain 1973] and the University of British Columbia [Starr & Mackworth 1978], has demonstrated the advantages of several methods for including contextual information. Growing small regions of similar spectral signature and classifying the regions, rather than individual points helps, as does the use of even rather primitive textural measures [Bajcsy 1973]. Amongst several other promising ideas are those recently expounded at the University of Maryland [Rosenfeld 1977b, Rosenfeld 1978] regarding the use of relaxation methods, also known as cooperative computation methods. While all of these methods produce results far superior to those generated by the point by point methods, it must be admitted that they require considerably more computing power.

#### *Line-finding*

As a last example we may look at edge-detection and line-finding. Many fast systems, some even running at full video speeds [Nudd 1978], use simple operations such as Robert's gradient, discrete approximations to the Laplacian operator or Sobel gradients. These produce visually pleasing results, but the edge fragments produced tend to be too noisy and ill-defined to succumb to concerted efforts to glue them into reasonably continuous lines and well-defined vertices.

Systems which do produce usable symbolic edge information such as those developed at M.I.T. [Griffith 1970, Horn 1971, Shirai 1975, Marr 1976, Marr 1978] require vast amounts of computing power both in terms of storage and machine cycles. Very similar sorts of things can be said about approaches which depend on scene segmentation using region growing techniques instead of edge-finding

[Brice & Fennema 1970, Ohlander 1975, Tenenbaum & Barrow 1977].

## VISION HARDWARE

Part of the explanation for this dichotomy then lies in the impatience of the implementers and the real world need for solutions to pressing problems involving processing of visual information. It is natural to think in terms of special purpose hardware suited to particular algorithms. For specialized tasks it is possible to realize one to three orders of magnitude speed-up in processing with affordable special purpose devices. In the past the development of such hardware was perhaps inappropriate since no one had enough confidence in any particular scheme to commit resources to an implementation effort. Also, the existence of fast systems that use very simple methods has discouraged further work, since "the problem has been solved." A look at the results quickly convinces one that this is not so.

## ROOTS

Several fields may be identified as having contributed major ideas to machine vision. I will single out just three here for discussion.

- (1) Image Processing
- (2) Pattern Recognition
- (3) Scene Analysis

Each field has now matured sufficiently to have its basic tools documented in a number of monographs, collections and text books [Andrews 1970, Biberman 1973, Gonzalez & Wintz 1977, Huang 1975, Lipkin & Rosenfeld 1970, Rosenfeld 1969b, Rosenfeld 1976b], [Cheng 1968, Fu 1974, Fu 1976, Grasselli 1969, Tou & Gonzalez 1974, Watanbe 1969], [Duda & Hart 1973, Hanson & Riseman 1978, Winston 1969, Winston 1977], as well as hundreds of papers. Indeed, I cannot begin to do justice to these here, but instead refer the reader to A. Rosenfeld's excellent bibliographies issued annually [Rosenfeld 1969a, Rosenfeld 1972, Rosenfeld 1973, Rosenfeld 1975, Rosenfeld 1976a, Rosenfeld 1977a]. In order to see where we are and to discern possible future trends we should analyze the strength and weaknesses of each of these paradigms in tackling the basic task we have set out for a machine vision system.

### *Image processing*

Image processing, as the name suggests, is something one does with images. Herein lies both its strength and its weakness. Of the three fields mentioned, this is the only one which deals with images as input. Indeed the basic operations apply to arrays of raw image intensities. Many useful transfers of ideas to machine vision can be traced to this emphasis. Un-

fortunately, image processing also produces images as output—not descriptions. Only in so far as these new, possibly enhanced, smoothed or sharpened images are easier to process are these techniques useful. Since, in the case of image processing, the final product is intended for human viewing, this is rarely the case. Further, one finds an unfortunate emphasis on linear, shift-invariant methods and consequently assorted transform techniques. Such ideas have only played a limited role in machine vision.

### *Pattern recognition*

At the core of this field is a method, pattern classification, which is concerned neither with images nor with descriptions thereof. Pattern classification instead deals with the mapping of vectors into integers—the vectors having components which represent measurements of some entity, the integers denoting the classes to which this entity might belong. This paradigm of feature extraction followed by pattern classification is of interest here, however, because many of its applications have involved features extracted from visual data. Several techniques used in the calculation of the numerical feature values for the classification process have found other applications in machine vision. Much of the sophisticated mathematical paraphernalia used to analyze the pattern classification stage has not.

### *Scene analysis*

Scene analysis concerns itself with the processing of descriptions of images into more sophisticated, or perhaps more useful, descriptions. In this category one finds much of the blocks-world work on line drawings [Roberts 1965, Clowes 1971, Huffman 1971, Waltz 1975]. As it turns out, obtaining the line drawings in the first place from the raw image information was the more difficult task; in fact, no system produces the perfect descriptions needed by early scene analysis systems [Winston, 1972, Grape 1972, Falk 1972].

More recently, discouraged by the complexity of the distributions of raw image intensities, researchers have turned to methods which exploit prior knowledge about the likely contents of the scene being viewed [Reddy et al. 1973, Tenenbaum & Barrow 1976]. In Max Clowes' words: "Vision is controlled hallucination." The image contributes a small "controlling" influence on the vision system's "hallucinations" based on expectations and predictions. Similar ideas have taken hold in other areas such as speech, where researchers despair of dealing with the complexities of the raw acoustic waveform without guidance from various "knowledge sources." There is however an ever-present danger of "controlled hallucination" turning into "hallucination." I think we may have closed our eyes to the raw image for too long.

## THINGS TO AVOID

The early years of any field tend to be characterized by a wide variety of approaches, many false starts and techniques based on inappropriate analogies. We can learn from these mistakes if we wish. Here are some things to avoid:

- (1) Using a mechanism-oriented approach, instead of a problem-oriented one.
- (2) Applying a known bag of tricks from another field.
- (3) Believing that complexity will automatically give rise to interesting behavior.
- (4) Hoping that "learning" will provide a boot-strapping mechanism.
- (5) Believing what works in a simple situation can be easily extended to a more complex one.
- (6) Suffering from theorem-envy—introducing unwarranted mathematical hair.
- (7) Working only on the "interesting" sub-problem—often not the weakest link.
- (8) Following the latest fad. Create your own instead!
- (9) Taking a random path through a maze of possibilities without explanation.
- (10) Admiring the King's new clothes.

## CURRENT TRENDS

Attempts are being made to apply machine vision methods to so many different problems using so many different methods that it is impossible to give any kind of coherent summary. Furthermore, progress is being made in understanding several important fundamental issues which cut across the spectrum of applications domains. It seems appropriate to concentrate attention to some of these issues.

### *Representation of objects*

If the task of the vision system is to produce useful descriptions of the scene being viewed, it is naturally important to pick a good representation for three-dimensional objects. If such a description is then to be used for recognition or in the determination of an object's position and orientation, it must capture information about the shape of the object and its disposition in space. This is an important problem, which does not occur in the processing of two-dimensional patterns such as microscopic image of bio-medical interest or in other areas such as finger-print identification or character recognition. A number of representations are currently being explored. One uses generalized cylinders or cones to approximate parts of objects after segmenting them into suitable pieces [Agin & Binford 1973, Nevatia 1974, Binford 1971a, Hollerbach 1976, Nevatia & Binford 1977, Marr & Nishihara 1977]. The information needed to construct such representations may be obtained by a variety of techniques including laser range finding [Nitzan et al. 1977] and stereo disparity calculations.

### *Spines*

Another representation of the shape of an object uses surface normals or "spines." This was suggested [Horn 1977], as a more appropriate representation than one in terms of elevations above some reference plane [Horn 1975a], in part because surface normals undergo a simpler transformation under rotation. Indeed, human performance on shaded images suggest that we are rather poor at establishing relationships in elevation, but have a pretty good idea about the local surface orientation. Fortunately, methods for determining this kind of information exist, ranging from photometric stereo [Woodham 1977] to the shape from shading algorithm. More recently this representation has been suggested as a half-way step to representation in terms of generalized cones [Marr 1978].

### *Early symbolic description*

From the discussion of the roots of machine vision it must be clear to the reader that the crucial thing missing from all three ancestor fields is the lack of a method which takes one from raw image intensities to symbolic descriptions. Little thought had been given even to the problem of where the appropriate point for this transformation would be. Recent work suggests that the first symbolic description be obtained at an early stage [Marr 1976] of the processing of the visual information. That is, the initial symbolic description contains very many items, each of a rather simple nature. Further analysis is then carried out using symbolic information processing techniques on this initial data base.

This is a considerable departure from vision work in the blocks world, where the first real symbolic description was a complete line drawing. Even then it was clear that this was inappropriate, and crude symbolic description and the mechanisms for manipulating them, existed hidden in huge assembly language programs [Horn 1971].

Many of the ideas regarding the use of early symbolic descriptions have come from a better understanding of human vision. Conversely, computer implementations provide an outstanding way of testing emerging theories about visual perception [Marr 1978]. Without such checks speculation runs rampant.

## UNDERSTANDING IMAGE FORMATION

It is not uncharacteristic of computer science to tackle a new domain with total disdain for the details of the mechanisms evident in that domain. Of more interest to the computer scientist are questions of computational structures and efficiency and whether a proposed algorithm will apply in the new domain. Machine vision is no exception in this regard. It seems that for a long time there was very little interest in the origins of the arrays of numbers given as input to a machine vision system. Recently it has been found that many constraints due to the physics of the real world situation can be successfully exploited, once understood [Waltz

1975, Horn 1975a]. This enables processing not otherwise possible.

Understanding the process of image formation is helpful in inverting the image formation. That is, it is useful to know how objects are imaged, if one wishes to build a symbolic description of what is being viewed from the image. This kind of consideration has focused attention on smooth variations in intensities in an image [Horn 1977, Horn 1978]. Previously, image intensities were processed only to extract regions of more or less uniform properties or to locate points of more or less rapid intensity change. At that point the image intensities themselves were discarded. This is unfortunate since a great deal of information about the objects being imaged is available there. This is quite different from the situation which applies in the case of binary images, useful in character recognition and printed circuit inspection, for example.

Basically, what one is after is information about the permanent properties of the objects, such as reflectance color and shape. This information is present in the raw image, but only in a coded fashion [Barrow & Tenenbaum 1978]. One may, for example, have to also deal with illumination conditions and shadowing. It is possible to extract all of this information from the raw image intensities, once the basic laws of image formation are understood. It is time to break the code.

## CONCLUSION

Progress has been made—at least we now know more about what we are up against. Much remains to be done. There is no shortage of good ideas right now, so we can discard some that no longer serve us well.

## REFERENCES

- Agin, G. J. and Binford, T. O. (1973), "Computer Description of Curved Objects," *Proc. 3rd Int. Joint Conf. on Art. Intell.*, Stanford University, Stanford.
- Andrews, H. C. (1970), *Computer Techniques in Image Processing*, Academic Press, N.Y.
- Arnold, R. D. (1978), "Local Context in Matching Edges for Stereo Vision," *Proc. DARPA Image Understanding Workshop*, May 1978, Baumann, L. (Ed), Science Applications, Inc.
- Bajcsy, R. (1973), "Computer Identification of Visual Surfaces," *Computer Graphics and Image Processing*, Vol 2, No. 2, pp 118-130.
- Barrow, H. G. and Tenenbaum, J. M. (1978), "Recovering Intrinsic Scene Characteristics from Images," in Hanson, A. & Riseman, E., *Computer Vision Systems*.
- Biberman, L. M. (Ed) (1973), *Perception of Displayed Information*, Plenum Press, N.Y.
- Binford, T. O. (1971a), "Visual Perception by Computer," *Proc. IFIP Conf.*, Dubrovnik, Yugoslavia.
- Binford, T. O. (1971b), "Visual Perception by Computer," *IEEE Conf. Systems and Control*, Miami, Dec 1971.
- Brice, C. and Fennema, C. (1970), "Scene analysis using regions," *Artificial Intelligence*, Vol 1, No. 3, pp 205-226.
- Clowes, M. B. (1971), "On seeing things," *Artificial Intelligence*, Vol 2, No. 1, pp 79-112.
- Cheng, G. C., et al. (Eds) (1968), *Pictorial Pattern Recognition*, Thompson Book Co., Washington D.C.
- Duda, R. O. and Hart, P. E. (1973), *Pattern Classification and Scene Analysis*, John Wiley & Sons, N.Y.
- Falk, G. (1972), "Interpretation of imperfect line data as a three-dimensional scene," *Artificial Intelligence*, Vol 4, No. 2, pp 101-144.
- Fu, K. S. (1974), *Syntactic Methods in Pattern Recognition*, Academic Press, N.Y.
- Fu, K. S. (Ed) (1976), "Digital Pattern Recognition, Springer, N.Y.
- Gennery, D. B. (1977), "A Stereo Vision System for an Autonomous Vehicle," *Proc. 5th Int. Joint Conf. Art. Intell.*, Cambridge, Mass.
- Gonzalez, R. C. and Wintz, P. (1977), *Digital Image Processing*, Addison-Wesley, Reading, Mass.
- Grape, G. R. (1973), "Model Based (Intermediate-Level) Computer Vision," Stanford Univ. A.I. Memo 201.
- Grasselli, A. (Ed) (1969), *Automatic Interpretation and Classification of Images*, Academic Press, N.Y.
- Griffith, A. K. (1970), *Computer Recognition of Prismatic Solids*, M.I.T. Project Mac, TR-73, M.I.T., Cambridge, Mass.
- Gupta, J. N., Kettig, R. L., Landgrebe, D. A. and Wintz, P. A. (1973), "Machine boundary finding and sample classification of remotely sensed agricultural data," *Machine Processing of Remotely Sensed Data*, Purdue University, 4B25-4B35.
- Hanson, A. and Riseman, E. (Eds) (1978), *Computer Vision Systems*, Academic Press, N.Y.
- Hollerbach, J. (1976), "Hierarchical Shape Description of Objects by Selection and Modification Prototypes," M.I.T. A.I. T.R. 346.
- Horn, B. K. P. (1971), "The Binford-Horn Line-Finder," M.I.T. A.I. Memo 285.
- Horn, B. K. P. (1974), "Determining Lightness from an Image," *Computer Graphics and Image Processing*, Vol. 3, No. 1, pp 277-299.
- Horn, B. K. P. (1975a), "Determining Shape from Shading," Chapter 8, in Winston, P. H. (Ed) *The Psychology of Computer Vision*.
- Horn, B. K. P. (1975b), "A Problem in Computer Vision: Orienting Silicon Integrated Circuit Chips for Lead Bonding," *Computer Graphics and Image Processing*, Vol 4, No. 3, pp 294-303.
- Horn, B. K. P. (1977), "Understanding Image Intensities," *Artificial Intelligence*, Vol. 21, No. 11, pp 201-231.
- Horn, B. K. P. and Bachman, B. L. (1978), "Using Synthetic Images to Register Real Images with Surface Models, C.A.C.M. (in the press).
- Hueckel, M. H. (1971), "An operator which locates edges in digital pictures," *J. Assoc. Comp. Mach.*, Vol 18, pp 113-125.
- Hueckel, M. H. (1973), "A local visual operator which recognizes edges and lines," *J. Assoc. Comp. Mach.*, Vol 20, pp 634-647.
- Huffman, D. A. (1971), "Impossible Objects as nonsense sentences," in *Machine Intelligence*, Vol 6, Mettler B. & Michie, D. (Eds), Edinburgh University Press, Edinburgh, pp 295-323.
- Huang, T. S. (Ed) (1975), *Picture Processing and Digital Filtering*, Springer, N.Y.
- Landgrebe, D. A. (1973), "Machine Processing for Remote Acquired Data," LARS Information Note 031573. Laboratory for Applications of Remote Sensing, Purdue Univ., Lafayette, Ind.
- Landgrebe, D. A. (1975), "NASA Contract NAS9-1416 Final Report," Laboratory for Applications of Remote Sensing, Purdue Univ., Lafayette, Ind.
- Lipkin, B. S. and Rosenfeld, A. (Eds) (1970), *Picture Processing and Psychopictorics*, Academic Press, N.Y.
- Mackworth, A. K. (1973), "Interpreting pictures of polyhedral scenes," *Artificial Intelligence*, Vol 4, pp 121-138.
- Marr, D. (1974), "A note on the computation of binocular disparity in a symbolic, low-level visual processor," M.I.T. A.I. Memo 327.
- Marr, D. (1976), "Early processing of visual information," *Phil. Trans. Roy. Soc. B 275*, 483-524.
- Marr, D. & Poggio, T. (1976), "Cooperative computation of stereo disparity," *Science* 194, 283-287.
- Marr, D. and Poggio, T. (1977), "A Theory of Human Stereo Vision," M.I.T. A.I. Memo 451.
- Marr, D. and Nishihara, H. K. (1977), "Representation and recognition of the spatial organization of three-dimensional shapes," *Proc. Roy. Soc. B* (in the press).
- Marr, D. (1978), "Representing visual information," in *Computer Vision Systems*, Hanson, A. & Riseman, E. (Eds).

44. Nevatia, R. (1974), "Structured Descriptions of Complex Curved Surfaces and Visual Memory," Stanford University, A.I. Memo 250.
45. Nevatia, R. (1976), "Depth Measurement by Motion Stereo," *Computer Graphics and Image Processing*, Vol 5, No 2., pp 203-214.
46. Nevatia, R. and Binford, T. O. (1977), "Description and Recognition of Curved Objects," *Artificial Intelligence*, Vol 8, No. 1, pp 77-98.
47. Nitzan, D., Brain, A. E. and Duda, R. O. (1977), "The measurement and use of registered reflectance and range data in scene analysis," *Proc. IEEE*, Vol 65, No. 2, pp 206-220.
48. Nudd, G. R., Nygaard, P. A. and Thurmond, G. D. (1978), "Charge-Coupled Device Technology for Smart Sensors," *Proc. DARPA Image Understanding Workshop*, May 1978, Baumann, L. (Ed), Science Applications, Inc.
49. Ohlander, R. B. (1975), "Analysis of Natural Scenes," Carnegie-Mellon University, Department of Computer Science, Ph.D. Thesis.
50. Quam, L. H. (1971), "Computer Comparison of Pictures," Stanford University A.I. Memo 144.
51. Reddy, D. R., Erman, L. D., and Neely, R. B. (1973), "A model and a system for machine perception of speech," *IEEE Trans. Audio and Electroacoustics*, AU-21, Vol 3, pp 229-238.
52. Roberts, L. G. (1965), "Machine Perception of Three-Dimensional Solids," in *Optical and Electro-optical Information Processing*, Tippet, J. T., et al. (Eds), M.I.T. Press, Cambridge, Mass.
53. Rosenfeld, A. (1969a), "Picture Processing by Computer," *Computing Surveys*, Vol 1, pp 147-176.
54. Rosenfeld, A. (1969b), *Picture Processing by Computer*, Academic Press, N.Y.
55. Rosenfeld, A. (1972), "Picture processing: 1972," *Computer Graphics and Image Processing*, Vol 1, pp 394-416.
56. Rosenfeld, A. (1973), "Progress in picture processing: 1969-71," *Computing Surveys*, Vol 5.
57. Rosenfeld, A. (1975), "Picture Processing: 1974," *Computer Graphics and Image Processing*, Vol 4, No. 2, pp 133-155.
58. Rosenfeld, A., Hummel, R. A. and Zucker, S. W. (1976), "Scene Labelling by Relaxation Operations," *IEEE Transactions on Systems, Man and Cybernetics*, SMC-6, pp 420-433.
59. Rosenfeld, A. and Kak, A. C. (1976), *Digital Picture Processing*, Academic Press, N.Y.
60. Rosenfeld, A. (1976a), "Picture Processing: 1975," *Computer Graphics and Image Processing*, Vol 5, No. 2, pp 215-237.
61. Rosenfeld, A. (Ed) (1976b), *Digital Picture Analysis*, Springer, N.Y.
62. Rosenfeld, A. (1977a), "Picture Processing: 1976," *Computer Graphics and Image Processing*, Vol 6, No. 2, pp 157-183.
63. Rosenfeld, A. (1977b), "Iterative Methods in Image Analysis," *Proc. IEEE Conf. on Pattern Recognition and Image Processing*, June 1977, pp 14-18.
64. Rosenfeld, A. (1978), "Some Recent Results Using Relaxation-Like Processes," *Proc. DARPA Image Understanding Workshop*, May 1978, Baumann, L. (Ed), Science Applications, Inc.
65. Shirai, Y. (1975), "Analyzing Intensity Arrays using Knowledge about Scenes," Chapter 2, in Winston, P. H. (Ed) *The Psychology of Computer Vision*.
66. Starr, D. W. and Mackworth, A. K. (1978), "Exploiting Spectral, Spatial and Semantic Constraints in the Segmentation of LANDSAT Images," University of British Columbia, Computer Science T.R. 78-1.
67. Swain, P. H. (1973), "Pattern Recognition: A Basis of Remote Sensing Data Acquisition," LARS Information Note 111572. The Laboratory of Applications of Remote Sensing, Purdue Univ., Lafayette, Ind.
68. Tenenbaum, J. M. and Barrow, H. G. (1976), "IGS: a paradigm for integrating image segmentation and interpretation," in *Pattern Recognition and Artificial Intelligence*, Academic Press, N.Y.
69. Tenenbaum, J. M. and Barrow, H. G. (1977), "Experiments in interpretation-guided segmentation," *Artificial Intelligence*, Vol 8, No 3, pp 241-274.
70. Tou, J. T. and Gonzalez, R. C. (1974), *Pattern Recognition Principles*, Addison-Wesley, Reading, Mass.
71. Waltz, D. L. (1975), "Understanding Line Drawings of Scenes with Shadows," Chapter 2, in Winston, P. H. (Ed) *The Psychology of Computer Vision*.
72. Watanabe, S. (1969), *Methodologies of Pattern Recognition*, Academic Press, N.Y.
73. Winston, P. H. (1972), "The M.I.T. Robot," *Machine Intelligence 7*, Edinburgh University Press, Edinburgh.
74. Winston, P. H. (Ed) (1975), *The Psychology of Computer Vision*, McGraw-Hill, N.Y.
75. Winston, P. H. (1977), *Artificial Intelligence*, Chapter 8, Addison-Wesley, Reading, Mass.
76. Woodham, R. J. (1977), "A cooperative algorithm for determining surface orientation from a single view," *Proc. 5th Int. Joint Conf. on Art. Intell.*, M.I.T., Cambridge, Mass.

# Image understanding architectures\*

by GRAHAM R. NUDD

*Hughes Research Laboratories*  
Malibu, California

## I. INTRODUCTION

One of the more complex processing problems facing both the military and commercial world is that of image analysis and understanding. The range of applications in which a real time computer vision system could have impact is extraordinarily large. In the military arena the need and potential benefits of a real time capability for target acquisition, autonomous guidance and image interpretation are well recognized and understood. To this end the Department of Defense is currently supporting several projects such as the DARPA Image Understanding (I.U.) Program to analyze both the processing requirements and machine organization for complex image processing and analysis. Truly autonomous systems which can perform both the imaging and processing in real-time using inexpensive and compact hardware will have significant impact in the tactical scenario for applications requiring remote sensing and analysis. (The term 'smart-sensors' has been introduced for hardware which performs both the sensing and processing in a single substrate.) In addition, an equally important impact might be made in areas such as automated production and inspection for machine assembly, etc. These two general areas are in many ways analogous and much of the hardware and software developed can be applied to both problems. Significant differences do, however, exist in terms of the signal to noise specification of the images, the required response time of the machinery, and the system constraints such as machine size and cost.

The image understanding problem can be divided into two largely separate issues. The first is the selection of the appropriate processing functions required to analyze the imagery, and the second is concerned with specification and design of the optimum machinery to perform those functions. The first issue has been the subject of much research traditionally performed on large scale general purpose machines at both academic institutes and industrial companies. Despite the very significant advances made in this area in recent years, this topic continues to be one of very fruitful research,

and while it is fair to say there is as yet no widely accepted general consensus concerning the optimum set of processing algorithms, a fairly well defined class of functions and instructions can now be enumerated. The second issue concerns the machine architecture and design to effectively perform the desired functions with sufficient throughput to enable real-time implementation. This paper is concerned primarily with this issue as it relates to present and emerging technology.

To set a perspective for this work, in Section II of this paper, we briefly review some architectural concepts for the machines developed to date and describe their performance goals. In Section III we discuss the type of arithmetic operations and instructions important to image understanding and give estimates of the potential throughput requirements. In this regard it is important to be aware of the distinction between required functions (such as convolution, for example) and the algorithm or technique required to perform it. This is significant with the very rapid changes in Very Large Scale Integrated circuit (VLSI) and Very High Speed Integrated circuit (VHSI) technology, since much of the algorithm development to date has been concerned with matching the necessary mathematical operations to current or previously developed machinery. In Section IV we describe the computational elements or primitives for the low level processing and discuss some of the techniques which may prove effective using both the present level of metal-oxide-semiconductor (MOS) integration and VLSI techniques. We illustrate this by some charge coupled device (CCD)/MOS circuits we have developed under the DARPA IU program, and discuss how these might be appropriate for a distributed architecture concept.

It should, however, be emphasized that the complexity of the imaging problem, both in terms of the processing requirements, algorithm definition and the necessary computation throughput, has not allowed a single unique solution for the optimum processor to be defined at this time. However, the advent of the new high density technologies provide the potential of removing one of the principal barriers to this solution, namely the conventional limits in throughput and processor complexity. Further, in many respects it appears that the issues of computer vision may well be one of the major beneficiaries of the new VLSI and VHSI technologies.

\* This work is supported in part by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Wright Patterson Air Force Base under Contract F-33615-76-C-1203, ARPA Order No. 3119.

## II. SOME EXISTING ARCHITECTURES

Computer vision and image understanding has been of interest for some time, and it is instructive to consider some of the machines designed and developed for this purpose.<sup>1-10</sup> Much of the effort to date in special purpose machinery for image understanding has concentrated on parallel architectures. It has been accepted for some time that sequential processors are not suited to computations where two-dimensional spatial relationships are an important part of the information. Moreover the necessary throughput, certainly at the low end of the processing, considerably exceeds that of conventional uniprocessors. For example, a sequential machine capable of  $1 \times 10^6$  instructions/sec. might take from several seconds to minutes to perform one relatively simple low level operation such as a  $5 \times 5$  convolution on an image with resolution equivalent is television.

The initial work on array machines for vision is typified by the work of Unger.<sup>1</sup> His concept was to have a central control providing instructions for the operation of a large rectangular array of identical modules (Figure 1). Each module was connected with its 4 nearest neighbors to allow data exchange, and consisted of an accumulator, a small amount of random access memory and some associated logic. One of the issues with this architectural concept is the complexity of the control and need for distributed storage. Also with the conventional discrete technology of that time the hardware became extremely complex as the array size increased. With the very high level of integration available with VLSI and VHLSI, configurations of this type could be more tractable. However, the concept of using an array of identical modules, with limited capability, working on local sub-areas, has formed the basis of much of the later work such as the ILLIAC III<sup>2</sup> which was aimed at the automatic analysis of binary images from bubble chambers. Its architecture was divided into three elements. The first performed operations for local pre-processing functions such as track thinning, gap filling, and line element recognition by an array of local processing elements (PE's) within a  $32 \times 32$  window. Then the manipulation and assembly of these line elements on a global

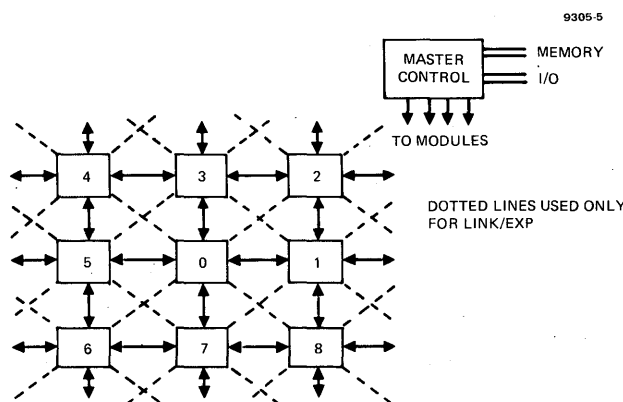


Figure 1—Parallel array processing concept developed by Unger (Ref. 1).

scale and the execution of higher-level mathematical analysis was performed in higher level machines. Data in each of the  $32 \times 32$  subwindows was processed sequentially so as to provide a line drawing of the binary patterns within the window. By labeling vertices, bends, crossovers and terminals, this line drawing was then converted into a linear graph (or list) structure and processed so as to characterize the local features. The machine itself suffered from the ability only to process binary images and the limited amount of processing that could be included in the local PE's. However, the partitioning of the processing into low-level local operations and the more sophisticated global operations is valid and with today's technology some of the limitations of the local PE's can be overcome.

Other parallel array architectures include the Cellular Logic Image Processor<sup>6</sup> (CLIP) which uses present generation n-MOS LSI technology to incorporate 8 identical PE's connected either as a rectangular or hexagonal lattice on a single chip. The master control is provided by a PDP 11/10. The logic structure is capable of performing simple Boolean operations, propagating operations, and bit-plane arithmetic operations. Simulations of this machine include line thinning, edge finding on grayscale images, solving maze problems and producing histograms and perimeters. The projections of the processing time for a single image are of the order of many seconds or minutes. PICAP, a modified version of the Parallel Picture Processing Machine (PPM),<sup>5</sup> first proposed by Kruse in 1973, consists of a mini-computer controlling both the processors and the image input and display devices. The PPM uses nine picture registers (each capable of storing a 4 bit  $64 \times 64$  pixel image) and two line buffers to obtain a  $3 \times 3$  processing kernel. Processing is performed by neighborhood matching logic which can accept inputs from the line buffers or directly from the picture registers, depending on whether it is a local operation or a pointwise comparison of multiple pictures. Templates are stored in the control unit for matching operations. The system has been in operation since 1975 and has been used for applications such as fingerprint coding, malaria parasite detection and printed circuit board inspection, but again the processing speed is significantly below "real-time rates." The execution time per operation for a  $64 \times 64$  pixel image is typically 2.5 ms, indicating a total processing time for the applications just mentioned varying from several seconds to minutes, depending on the complexity of the function to be performed.

A machine with well documented performance characteristics is the Toshiba Pattern Information Cognitive System (TOSPICS)<sup>7</sup> which is a distributed processor controlled by a TOSBAC-40C mini-computer, employing hardwired modules to perform the basic image processing functions. The seven primitives and their processing times are listed in Table I. By building on the basic image processing functions, more global and complex functions such as edge detection, texture analysis and shape identification can be performed. Processing of neighborhoods in the image is sequential, but data access and computation at each pixel position are performed in parallel. The use of hardwired special purpose

modules allows considerable speed increase but, as can be seen in Table I, the operations are approximately two orders or less than that required for real-time processing.

Other machines such as the ILLIAC IV,<sup>8</sup> STARAN E<sup>9</sup> and PEPE,<sup>10</sup> although not designed or matched specifically for image understanding, include processing functions of value in the imaging world.

While most of the above machines use a parallel array structure, the efforts to date have not resulted in a processor with sufficient throughput for real-time image analysis or understanding particularly for missions of interest to current military systems. Indeed, in some respects a highly parallel structure of identical PE's may introduce greater penalties than benefits. For example, the control and processing issues soon dominate the system. The interconnects, in general, grow as  $n^2$  (for an array of  $n$  elements). The reliability and maintainability of massively parallel structures even with LSI and VLSI is a significant problem. In Section III we discuss an alternate approach using a number of hardwired, special purpose circuits with sufficient throughput to process the low-level operators in real-time. This approach (similar to TOSPICS) is particularly well suited to the pre-processing or feature extraction level where a widespread unanimity exists on the required operations, such as edge detection, line thinning, line linking and the calculation of low-level statistics such as histograms, variances, etc. These processes then feed into a more general purpose machine for the higher level (or symbolic) type operations where the probability of branching is high but the required throughput has been reduced by several orders of magnitude.

### III. PROCESSING REQUIREMENTS FOR I.U. MACHINES

It is well recognized that the processing requirements for real-time image analysis systems, particularly for present and proposed military missions, greatly exceed that of currently available general purpose computers. As an illustra-

tion of this, in Figure 2 we show estimates of both the throughput and memory requirements for a number of military systems under consideration. From this it is clear that some form of special machinery will be needed to approach these capabilities. Further, many of the basic assumptions used in present and previous generations of processors may not be valid in the VLSI technology era. A significant example of this is the hitherto paramount aim to reduce the necessary gate count in the processor so as to reduce the machinery cost and increase the reliability. This constraint may well be invalid in future special purpose processors where the gate density may exceed  $10^5$ /chip. A good case has been made that the data manipulation and interconnects may be the significant burdens in future machines where gates are essentially free and Vias cost highly in terms of silicon area, delay times and design effort.<sup>11</sup>

An alternative to the highly parallel configuration of identical elements shown in Figure 1 can be devised if the processing is divided into distinct regions (or levels) determined by factors such as the required throughput, probability of branching and word length. When this is done, the "low-level" processing (directly at or adjacent to the sensor) can be characterized by very high throughput, low branching probability and relatively low accuracy requirements.

At this end of the computation the throughput is high because typically each picture element must be processed (in combination with its local neighbors) usually in terms of a rather simple operation such as local averaging or convolution. Typically the kernel size for the low-level operations will range from  $3 \times 3$  to  $15 \times 15$  pixels. This requires, for linear operations such as convolution and spatial filtering, an instruction rate of the order of

$$\text{MIPS} \cong (n_1 \times n_2) \times I \times O \times N^2 \times F \times 10^{-6} \quad (1)$$

where the kernel size is  $n_1 \times n_2$  pixels

$I$  is the number of instructions per operation  
 $O$  is the number of operations in the algorithm  
 $N$  is the number of pixels/line and  
 $F$  is the frame rate

TABLE I.—The Basic Image Processing Functions Performed by TOSPICS and Execution Times

Function	Applications	Execution Time
Two-Dimensional Convolution	Laplacian, Smoothing, etc.	262 mS
Logical Filter	Feature Detection Thinning	262 mS 262 x mS
Region Labeling	Region Separation Particle Measurement	524 mS 786 mS
Data Conversion	$\gamma$ - Correction Thresholding	262 mS 262 mS
Histogram Generation	Frequency Counting	262 mS
Affine Coordinate Transform	Enlargement with Rotation Axis Skewing	262 mS 262 mS
Pixel Operation	Logical Operation Arithmetic Operation	524 mS 524 mS
Gradient Operation	Edge Detection	2.36 S

For local averaging over a  $3 \times 3$  pixel array on the image of television quality we obtain a throughput requirement, for just this function, of approximately 100 million instructions/sec. For the next level of complexity, such as edge detection, the required instruction rate is typically in excess of 500 million instructions/sec. Immediately we can see that this is far in excess of the capability of current machines typically operating at 1-10 MIPS. The situation is made somewhat worse when low-level statistical operations such as histogramming or variance calculation requiring operations proportional to  $(n_1 \times n_2)^2$  are considered. In this case throughputs in excess of 1,000 MIPS are required for even relatively small kernels. Fortunately, at this level of image understanding there is little need for conditional branching or high accuracy. Further, at the front-end or low-level a widely held consensus exists of the types of processing functions nec-



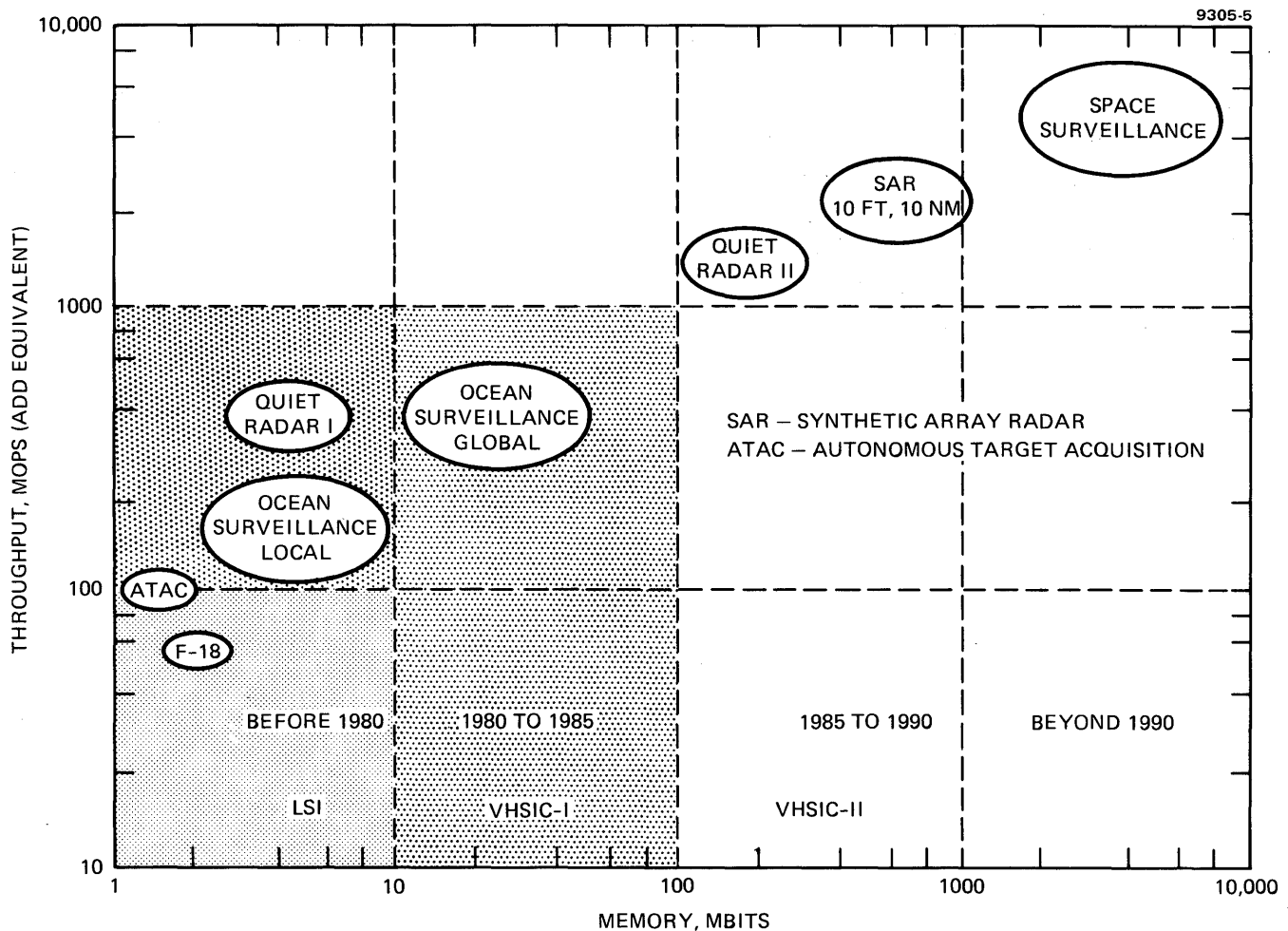


Figure 2—Estimation of processing and memory requirements for potential military systems.

essary; edge detection, edge linking, median filtering, histogramming, etc.

This is in contrast to the next level of the process which is highly image dependent and, as yet, a true consensus of the necessary operations has not been formed even among the I.U. experts. Typical operations required at the high end of the processing are symbolic manipulations of the lines and vertices extracted from the image by the low level architecture, and segmentation of the image using the features such as texture derived by the low level operators. Of paramount importance is the spatial relationship of the features (do the lines interconnect so as to suggest a road or building, etc.). The requirements at this stage can be characterized by: low throughput (the input data rate having been reduced by a factor of  $10^2$  to  $10^4$  from the raw pixel rate, by the extraction of lines for example), and relatively long word length representing the requirement for high accuracy. Fairly sophisticated operations can be envisioned perhaps involving the small difference of relatively large numbers and therefore the word length should be significantly greater than the 6 bits or so typically used to represent the raw pixels.

An illustration of a hypothetical but representative I.U. system is given in Figure 3. The overall processing requirement can be considered to be a triangle as shown with throughput represented in the horizontal direction and processing sequence represented vertically. The transition between the low and high levels typically occurs after the required throughput has been reduced from  $10^2$  MIPS or above to the order of 0.1 MIPS or less. Much of the data below the interface is in terms of processed pixels and the spatial relationship of the imagery is maintained by a pixel count whereas above it the data is primarily in symbolic form represented as lists, etc. The data reduction might typically be of the order of  $10^3$  or more.

These two distinct regions are best suited to different types of processors. At the low-end which might typically represent 80 percent or more of the processing, custom built special purpose primitives which perform specific functions such as convolution, edge detection or line linking in real-time seem most appropriate. The "high-level" processing might best be performed by a general purpose architecture, which could possibly be a commercial device if sufficient

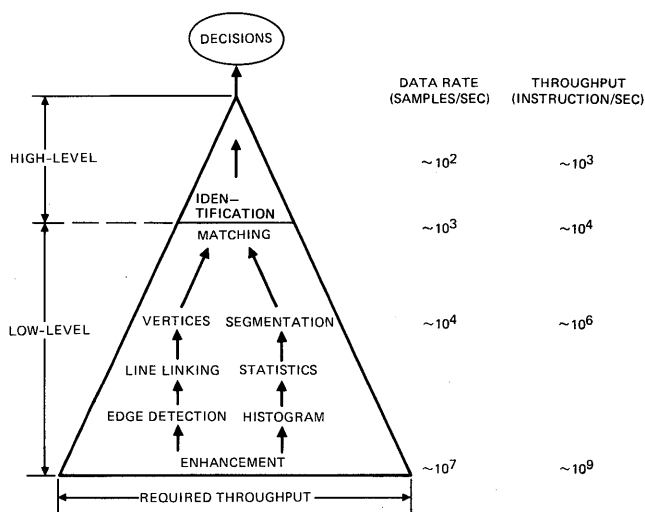


Figure 3—Processing requirements of typical image understanding system.

throughput can be achieved or a special purpose programmable device with a limited instruction set suited directly to the I.U. The resulting architecture shown in Figure 4a consists of a concurrent configuration of low-level or primitive processors each performing a single function such as convolution, edge detection or histogramming. These processors perform all the enhancement operations, the feature extraction and low level statistical operations. The output from these which will consist of linked lines, or boundaries of similar textures, etc., will be passed to the symbolic processor in the form of lists or vectors. This processor performs a wide variety of operations to match the features and determine the shape of objects and make decisions based on the required mission constraints. As an example, a configuration for a typical military mission is shown in Figure 4b. Three key issues are involved in configuring such a machine; the selection and design of the primitives, the control and local storage requirements, and the necessary instruction set of the high-level (symbolic) processor.

#### IV. SELECTION AND CONFIGURATION OF PRIMITIVES

Two essential demands are made on the primitives for the I.U. configuration discussed above. Firstly, they must form a full and comprehensive set of low-level operations for a wide variety of image types and configurations. Without this, the range and applicability of the machinery will be limited to a sub-set of the vision problem. (If the control unit is appropriately designed the addition of new primitives as required may be possible.) Secondly, each primitive should be capable of accepting input data at a rate equivalent to real-time imagery (taken here to be 7.5 MHz) and provide processed data at these rates. To determine the feasibility and possible configuration of the primitives, it is necessary to catalog the low-level operations and their instruction rate.

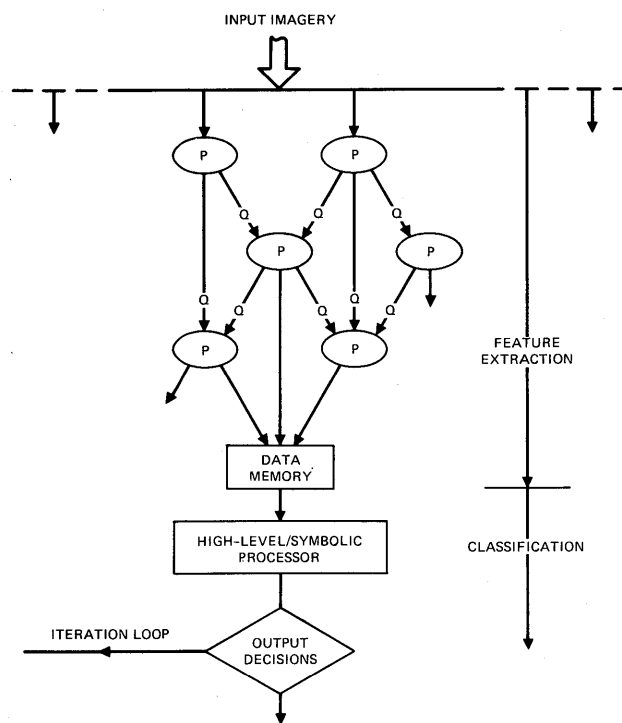


Figure 4(a)—Possible image understanding architecture.

A partial listing of some of the more common functions is given in Table II together with the required throughput and accuracy. Several distinct classes of function can be identified including convolutional type operations such as spatial filtering, local averaging and edge detection; low level statistical operations such as sorting histogramming, median and variance calculation; and logical operations such as binarization, edge thinning line linking, etc.

The formulation of some of these functions are described

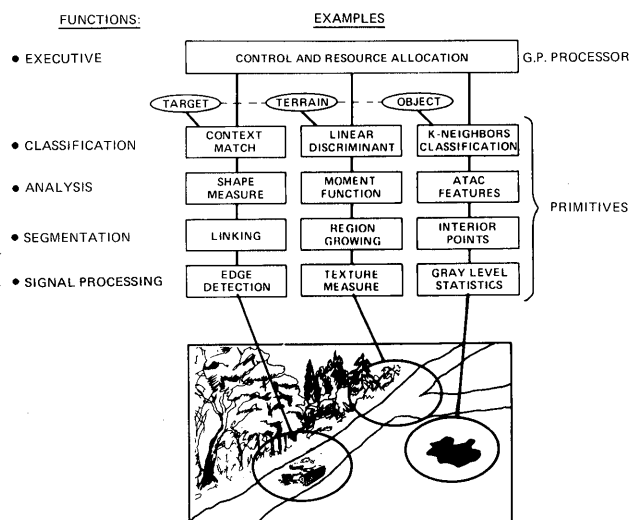


Figure 4(b)—Image analysis system for potential military application.

TABLE II.—Typical Low-Level Operations and the Processing Requirements

	Required Throughput	Required Accuracy (bits)
<i>Enhancement</i> 0(p)		
local averaging	} >10 <sup>2</sup> MIPS	6
non-linear gain		
<i>Feature Extraction</i> 0(p)		
spatial filtering	} >10 <sup>2</sup> MIPS	6
convolution		
edge detection		
<i>Statistical Operations</i> 0(p <sup>2</sup> )		
sorting	} >10 <sup>3</sup> MIPS	6
median operation		
histogram		
variances, etc.		
<i>Spatial Operations</i> ~0(p) × 10 <sup>-1</sup>		
thresholding	} <10 MIP	~1
thinning		
edge linking		
<i>Intermediate Operations</i> <0(p) × 10 <sup>-2</sup>		
vertex matching	} <0.1 MIP	≥16
moment calculation		
segmentation		

below using the three by three array of pixels shown in Figure 5.

The convolution operation consists of using a 3 × 3 template

$$W = \begin{bmatrix} W_{i-1,j-1} & W_{i-1,j} & W_{i-1,j+1} \\ W_{i,j-1} & W_{i,j} & W_{i,j+1} \\ W_{i+1,j-1} & W_{i+1,j} & W_{i+1,j+1} \end{bmatrix} \quad (2)$$

where  $W_{i,j}$ , etc., may be either positive or negative to weight the individual picture elements in the kernel (Figure 5) prior to summation. In this case the necessary multiplications and adds must be performed at the pixel rate (0.13 μsec) to form the full processed image

$$\bar{I} = I * W \quad (3)$$

in real-time. This is equivalent to a sequential rate of 135 MOPs. For the simplest case where all the weights  $W_{i,j}$ , etc., are unity the resulting output is proportional to the local average

$$\bar{I}(e) = \text{const}[a + b + c + d + e + f + g + h + i]. \quad (4)$$

For more complicated functions such as matched filtering, etc., the individual weights can have a full gray scale typically equivalent to 6 bits. For example edge detection can be performed by performing a variety of convolutions, one for each possible edge orientation. Using this approach horizontal and vertical edges might be found by using templates equivalent to

$$W = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix} \quad (5)$$

and

$$W = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix} \quad (6)$$

respectively. This approach can be generalized to find each edge component at angular increments of 45° using the 3 × 3. If finer resolution is required the kernel size must be increased. A specific edge calculation algorithm which can be performed as a convolution and has found widespread acceptance in the Sobel Operation<sup>12</sup>

$$S(e) = \frac{1}{8} [ (a + 2b + c) - (g + 2h + i) + (a + 2d + g) - (c + 2f + i) ]. \quad (7)$$

Two coincident templates

$$W_x = \frac{1}{8} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

and

$$W_y = \frac{1}{8} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad (8)$$

together with an absolute magnitude operation and addition are required for this. The Laplacian operator which performs a two dimensional differentiation equivalent to

$$\partial^2 I / \partial x^2 + \partial^2 I / \partial y^2$$

can be used either as a preprocessing operator<sup>12</sup> or for edge crispening. It requires a template equivalent to

$$W = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix} \quad (9)$$

The range of operations that can be performed by specialized or programmable convolutions in image understanding is extraordinarily large. In many applications as much as half of the total processing is either directly in the form of convolutions or can be expressed in these terms, and hence this operation is an essential element in the selection of the low level special purpose primitives.

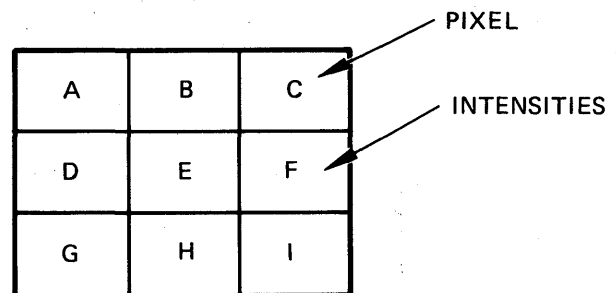


Figure 5—Notation used for 3 × 3 pixel kernel.

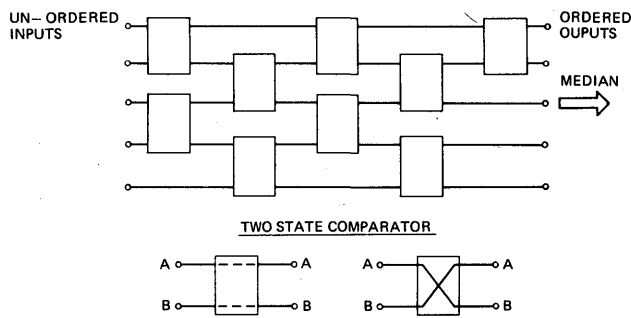


Figure 6—Concept of parallel sorter and median operator.

In addition the simple logical operations, such as absolute magnitude determination  $\bar{I}(e) = |I(e)|$ , binarization about some calculated parameter such as the local mean

$$I_b(e) = \begin{cases} 1 & \bar{I}(e) < e \\ 0 & \bar{I}(e) \geq e \end{cases} \quad (10)$$

as well as a variety of simple hardwired logic functions, are required at the low level. Examples of these occur in line thinning and line linking<sup>13</sup> which typically occurs after the edge detection. In general the throughput requirement for this type of operation is much less than the previous examples and these circuits can readily be accomplished with present MSI or LSI technology. More significant demands are made by even quite low order statistical operations such as histogramming sorting and median filtering. In general these operations take calculation times proportional to  $(n^2)$ .<sup>14</sup> The median calculation is illustrative of this class, and is widely used to eliminate noise spikes, thin edge elements and as a size filter. (The median operation of a collection of pixels  $I_1, I_2, \dots, I_n$  for  $n$  odd is that intensity for which  $(n-1)/2$  elements are smaller or equal in value.) The classical way to perform this is to perform a rapid sort of the picture intensities of the given array and simply take the  $[(n+1)/2]$  largest output. A simple sequential circuit<sup>15</sup> which performs this is real-time but with a latency of  $(n-1)/2$  cycles using  $(n-1)^2/2$  simple comparators is shown in Figure 6. This

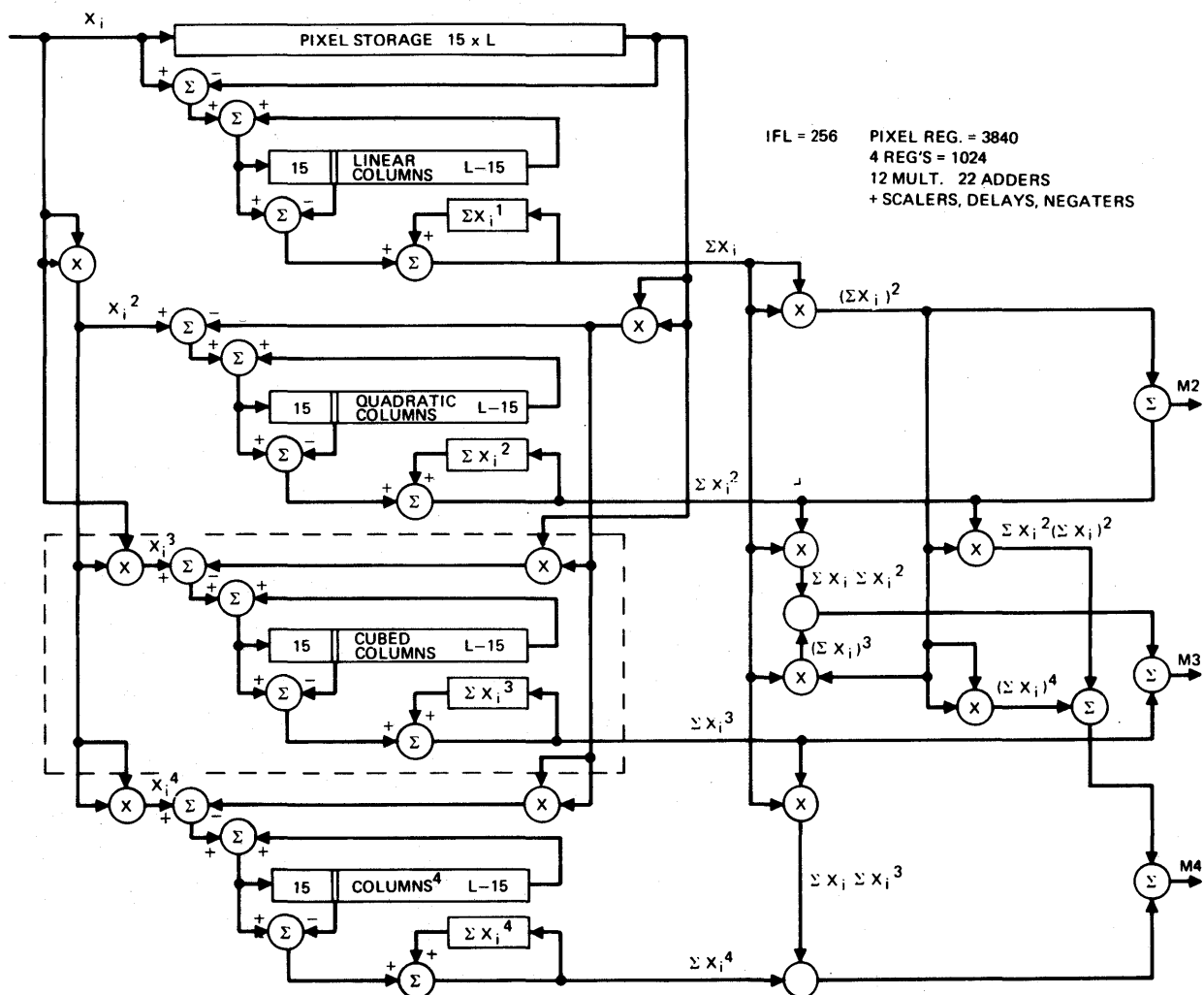


Figure 7—Architecture for moment calculation.

sorter can form the basic element of many other statistical operations such as histogramming, variance and mode filtering. The relatively high cost in terms of gate density to achieve the essentially parallel structure is appropriate to VLSI because of its high degree of regularity.

An additional operational which is frequently required is the moment calculation of the form

$$M_p = \sum (I_{ij} - \bar{I}_{ij})^p \quad (11)$$

where  $I_{ij}$  is the center picture intensity

$\bar{I}_{ij}$  is the local average

$n$  is the number of pixels in the kernel

$P$  typically varies from 1 to 5.

(Equation 11 provides the mean, variance, skewness and kurtosis for  $P=1, 2, 3$  and  $4$  respectively.) This is widely used in calculations for texture analysis and segmentation among other operations and is of sufficient interest to be included as a low-level primitive. If the calculation is performed directly the operation rate required for a  $15 \times 15$  array is in excess of  $10^3$  MOPs. A preferable approach is to calculate the non-centered moments

$$M_1 = \frac{1}{M} \sum (I_{ij}) \quad (12)$$

$$M_2 = \frac{1}{N} \sum (I_{ij})^2 \text{ etc.}$$

A possible architecture for doing this is shown in Figure 7.

## V. DESIGN AND PERFORMANCE EVALUATION OF SPECIAL PURPOSE PRIMITIVES

Under partial support of the DARPA Image Understanding Program we have been investigating techniques to perform these low-level functions at real-time rates.<sup>16,17</sup> As part of this program we have developed a number of custom designed CCD/MOS LSI circuits to demonstrate the feasibility of real-time operation. Initially our work was concerned with operation over a  $3 \times 3$  pixel kernel but more recently we have performed operations over  $5 \times 5$ ,  $7 \times 7$  and  $26 \times 26$  kernels using both fixed operators and voltage programmable devices. A list of the algorithms implemented is given in Table III. In the initial phase of this investigation we have concentrated on charge couple device (CCD) and MOS technology for two principal reasons. Firstly, since many of the next generation of military imagers will themselves be CCD's it is envisioned that many of the low-level operations can be incorporated directly at the focal plane as illustrated in Figure 8. Secondly, the power delay product (0.1 pico-Joules) of CCD technology is significantly below that of competing circuitry and hence the potential for integration is that much higher.

The technique used to date is essentially a sampled data approach and as such is probably best suited to accuracies

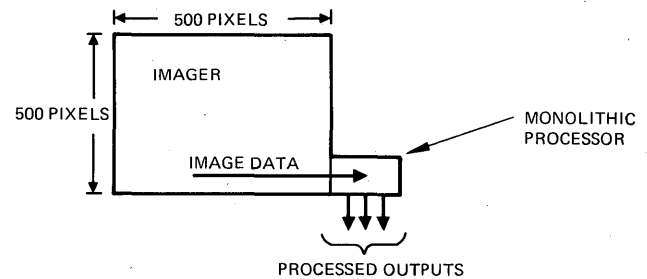


Figure 8—Concept of smart sensor for monolithic imager and processor.

of about six bits (which is sufficient for the low-level operators). It does, however, allow the necessary processing speeds for real-time operation and many of the architectural concepts can be translated to either binary operation or mixed radix multi-valued logic. The three test chips developed to date are shown in Figures 9-11. Where possible we have structured the processors in the form of a sampled data two-dimensional transversal filter to achieve optimum pipelining.

A photomicrograph of Test Chip I is shown in Figure 9. The area shown is equivalent to  $40 \text{ mils} \times 50 \text{ mils}$ , and the two-dimensional filter can be seen at the center. The circuit itself is an n-channel structure with a feature size equivalent to  $\sim 6 \mu\text{m}$ . The processing operations (edge detection, spa-

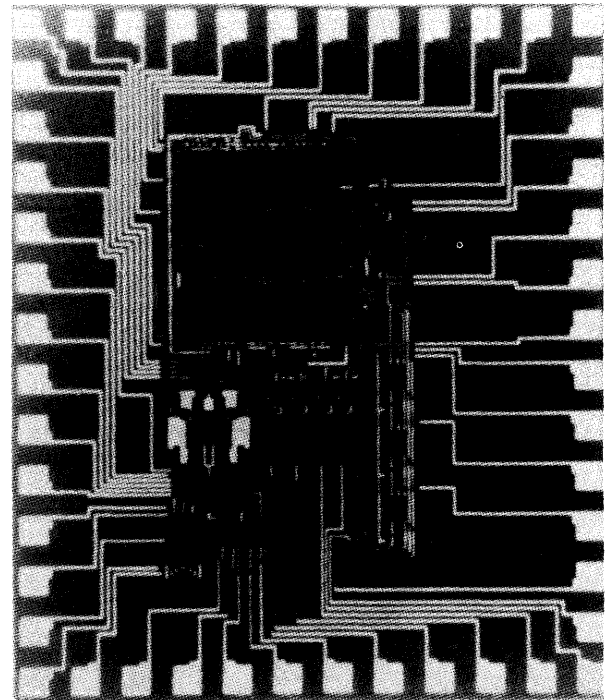


Figure 9—Photomicrograph of test chip I developed for the DARPA I.U. program.

TABLE III.—CCD/MOS Circuits Developed on the I.U. Program

Test Chip Numbers	Algorithms Implemented	Kernel Size	Operations per Pixel	Effective Operation Rate
I	Edge detection	3 x 3	16	80 KOPS
	High-pass spatial filter	3 x 3	18	90 KOPS
	Laplacian	3 x 3	13	65 KOPS
	12 dB/aperture corrector	3 x 3	18	90 KOPS
II	Sobel	3 x 3	16	32 MOPS
	Mean	3 x 3	9	18 MOPS
	Unsharp masking	3 x 3	13	26 MOPS
	Binarization	3 x 3	10	20 MOPS
	Adaptive stretch	3 x 3	12	24 MOPS
III	Laplacian	3 x 3	13	91 MOPS
	Mask programmable convolution	7 x 7	98	636 MOPS
	Programmable convolution	5 x 5	50	350 MOPS
	'Plus' shaped median	5 x 5	625	$\sim 10^3$ MOPS
	Bipolar convolution	26 x 26	1352	$\sim 10^4$ MOPS

tial filtering, and Laplacian and aperture correction) have all been pipelined to increase the throughput as shown in Figure 12. Examples of the processing capability for edge detection are shown in Figure 13. The image shown has a resolution of  $128 \times 128$  pixels with an intensity resolution of equivalent to 4 bits. In Figure 14 we show a real-time test facility which takes  $512 \times 512$  pixel data at 30 frames/sec directly from a CCD or vidicon camera and drives a single LSI chip which performs the functions described in Equations 13 through 17. The processed results are displayed directly on the monitor shown in real-time.

Edge Detection:

$$S(e) = \frac{1}{2} [ |(a+2b+c) - (g+2h+i)| + |(a+2d+g) - (c+2f+i)| ] \quad (13)$$

Local averaging:

$$f_m(e) = \frac{1}{8} [a+b+c+d+e+f+g+h+i] \quad (14)$$

Unsharp masking:

$$S_u(e) = (1-\alpha)e - \alpha f_m(e) \quad (15)$$

Adaptive binarization:

$$S_b(e) = \begin{cases} 1 & \text{for } f_m(e) \leq e \\ 0 & \text{for } f_m(e) > e \end{cases} \quad (16)$$

Adaptive stretch:

$$S_a(e) = \begin{cases} 2 \min e, r/2 & \text{for } f_m(e) \leq r/2 \\ 2 \max e, r/2, 0 & \text{for } f_m(e) > r/2, \end{cases} \quad (17)$$

where  $r$  is the maximum pixel intensity.

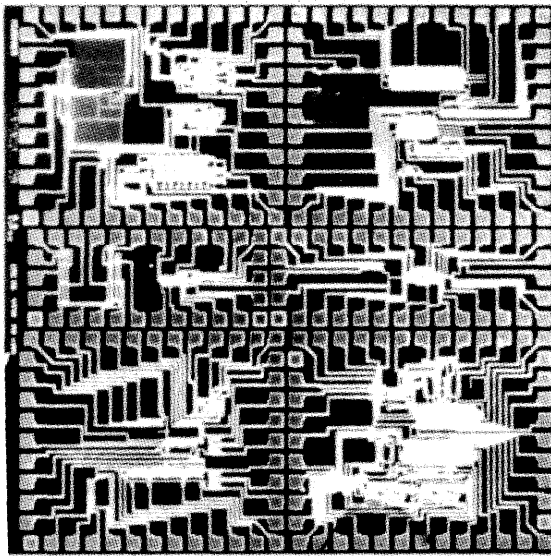


Figure 10—Test chip II developed for the I.U. program.

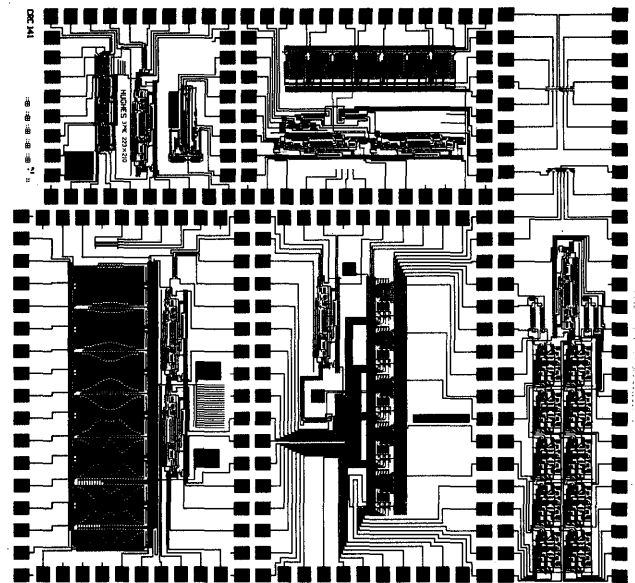


Figure 11—Test chip III.

Examples of the real-time processing capability of these primitives using data fed directly from a vidicon camera are shown in Figure 15.

Other primitives developed in this program include the  $5 \times 5$  programmable convolution array, a median filter/sorter circuit, and a large bipolar convolutional filter designed as

a pre-processor for the "primal sketch"<sup>18</sup> vision system. A photomicrograph of the programmable device (which is capable of changing the convolutional template of a  $5 \times 5$  array at frame rates (30 Hz)), together with three arbitrary impulse responses corresponding to differing sets of input weights

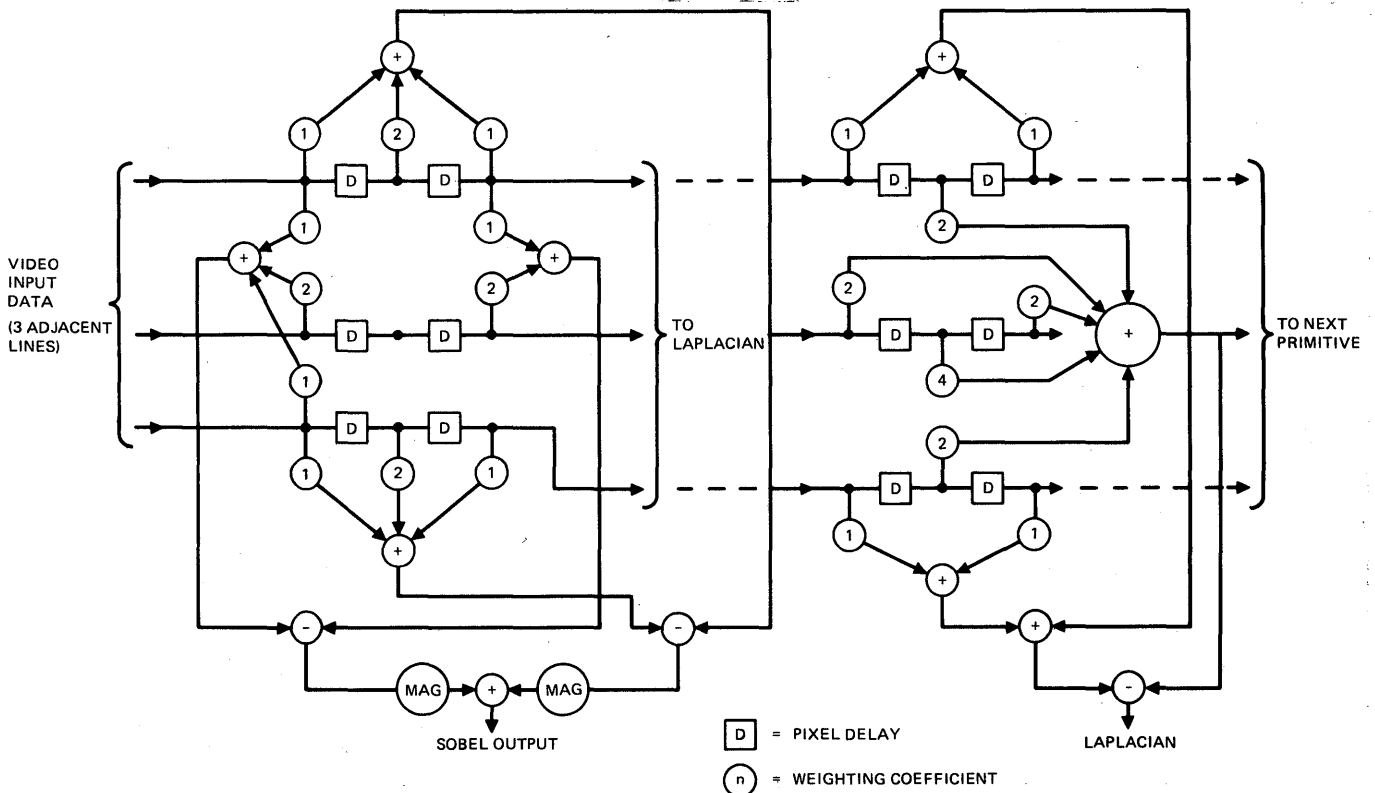


Figure 12(a)—Schematic of the pipeline concept used to configure the low-level operator.

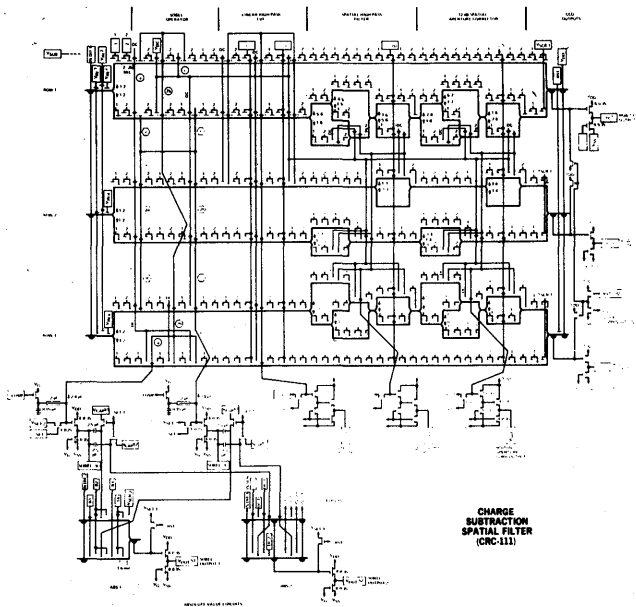


Figure 12(b)—Circuit schematic of test chip I.

is shown in Figure 16. The added complexity of providing programmability to this primitive is worthwhile due to the wide variety of convolutions employed. The median primitive, built using an architecture equivalent to that shown in Figure 6, is shown in Figure 17 together with the results when used both to extract noise equivalent single pixel blemishes and as a width filter. In this case the width or size filter is shown as extracting lines of two pixels or less width. This type of operation is widely applicable both as an enhancement tool and a screener or auto-cueing technique matched to objects of a predetermined size.

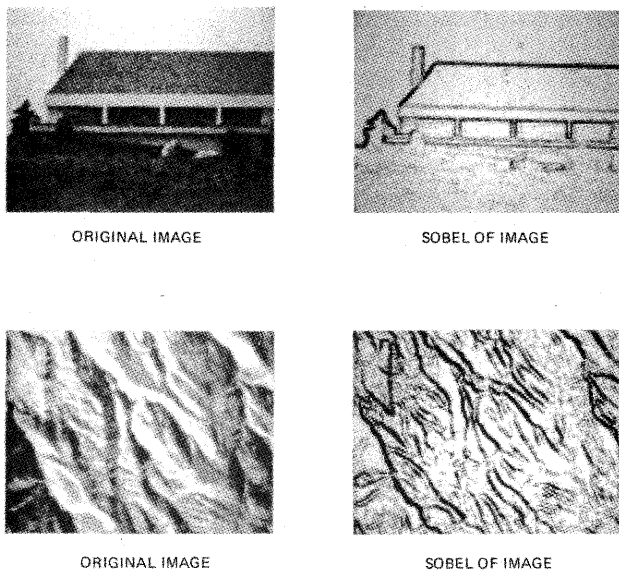


Figure 13—Example of edge detection performance using the CCD/MOS primitive.



Figure 14—Test system developed for real-time operation of the low-level functions.

Finally we show in Figure 18 a very large ( $26 \times 26$ ) bipolar convolution with the circularly symmetric weighting function illustrated, for the primal-sketch theory of vision. The effective operation rate in this device is of the order of  $10^4$  MOPs. This circuit is designed to be used in an image understanding system matched to the human eye and is a pre-processor for both edge detection and stereo analysis, based on the theories of human vision developed by Prof. Marr and his colleagues at the Artificial Intelligence Laboratories of M.I.T.

The above operations and circuits are representative of

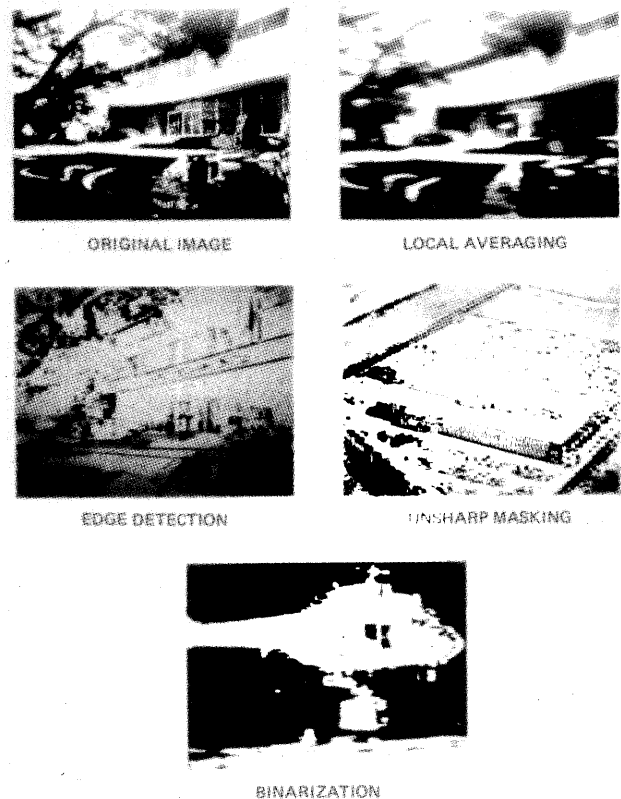
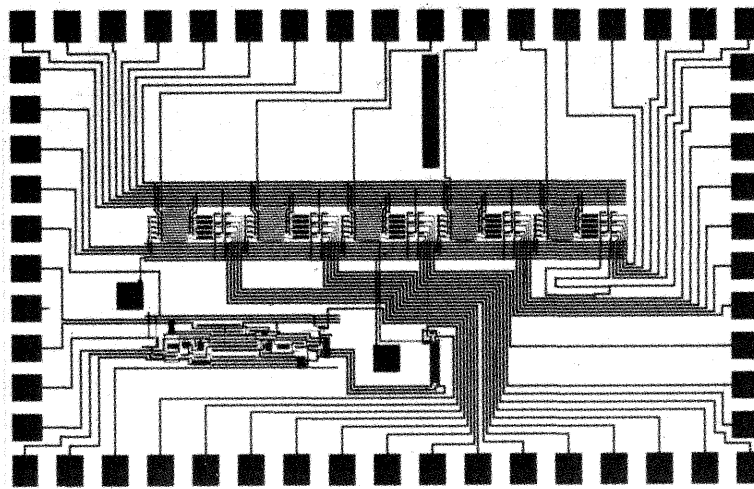
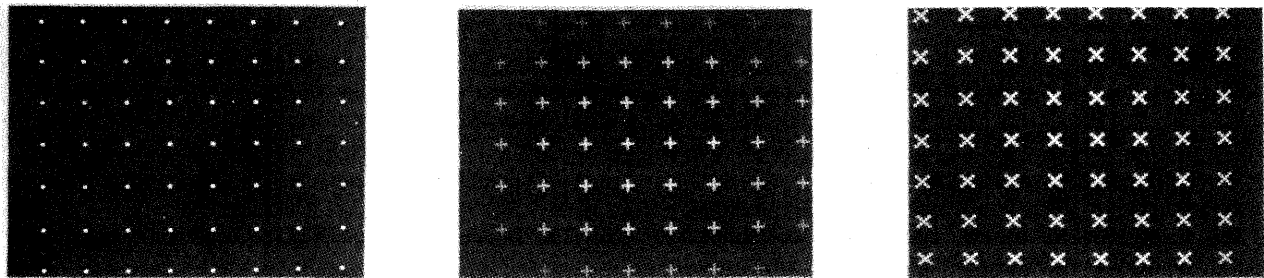


Figure 15—Example of real-time operation of test chip II.





(a)



(b)

Figure 16— $5 \times 5$  programmable convolution. (a) Circuit layout.  
(b) Programmed impulse response.

the high throughput requirements of the low-level primitives shown in Figure 4. From the experience gained in this work we feel it is indeed possible to build such primitives operating at real-time television rates. Although the sampled data techniques employed here are limited in accuracy to about six bits, the increased throughput available over conventional binary circuitry and the fact that most of the low-level operators require no more than six bit accuracy indicates that in many applications the approach may be appropriate. Probably an optimum situation, which we are currently pursuing, is a mixed radix operation which allows accuracy and throughput to be balanced throughout the system. At the low level where high throughput and limited accuracy is required, the radix or base might be quite high, say 8 or 16, whereas in the latter stages in the symbolic processor it would drop to the conventional two.

## VI. CONTROL AND INTEGRATION ISSUES

The control issues for the types of processor shown in Figure 4 are somewhat simplified with respect to the large lattices discussed in Section II, but still present very signif-

icant problems. Firstly, since each primitive carries an embedded knowledge of its processing algorithm, there are very limited issues concerning the instructions. At most these will take the form of changes in weighting functions and kernel sizes which probably occur very infrequently. The control issues center around storage of data to account for the differing latency in the primitives and the programmable data paths between them. Since each local processor is operated at the full pixel rate local storage can approach a minimum equal to the difference in latency. If the operations are performed in the inter-pixel period as discussed in Section V, this is limited to the difference in size between the largest and smallest kernels, which, however, might be quite large. For example, a range of from one pixel to an array of  $15 \times 15$  is not uncommon, in which case a local storage of 210 pixels ( $\sim 30 \mu\text{sec}$ ) has to be included in the worst case to provide a totally synchronous operation. It is worthwhile to note that this is comparatively simply achieved using the 2D CCD array approach by simply extending the number of CCD stages. But in general this is not so simple and considerable simplification might be made if the selection of primitives can be chosen 'a priori' and the data transfer latched. The data routing problem is also significant and

requires additional study. It is possible that an extension of the 'data-flow' concepts developed by Dennis et al.<sup>19,20</sup> may be applicable, where the routing is determined by appending a key to each data stream. The concept might be to have a series of comparators and exchange modules to route data similar to the modified Batcher circuit<sup>15</sup> shown in Figure 6.

Since most of the high throughput tasks will have been performed prior to the symbolic processor the architecture of this can conform to the conventional Von Nuemann concepts. Indeed a single commercial processor or series of micro-computers might be appropriate, or, as the technology develops it might be appropriate to develop a single high-throughput processor with a limited instruction set designed specifically for I.U.

With the advent of VLSI and VHSI with gate densities of  $10^5$ /chip and clock rates approaching  $10^8$  Hz we could expect much of the hardware for the full system to be put on relatively few chips. While this is ideal in terms of the pin-out limitations and avoiding the necessary drivers to get off the individual chips, the true benefits will be derived from an optimum partitioning. Effective chip design will require that the data bandwidth between packages be reduced to a minimum. Those primitives which operate on each pixel and produce a processed pixel for each clock cycle are the most significant data handlers and an optimum configuration would require as many of these on a single substrate as possible. To enable these new technologies to provide the maximum impact to the I.U. community, research must be di-

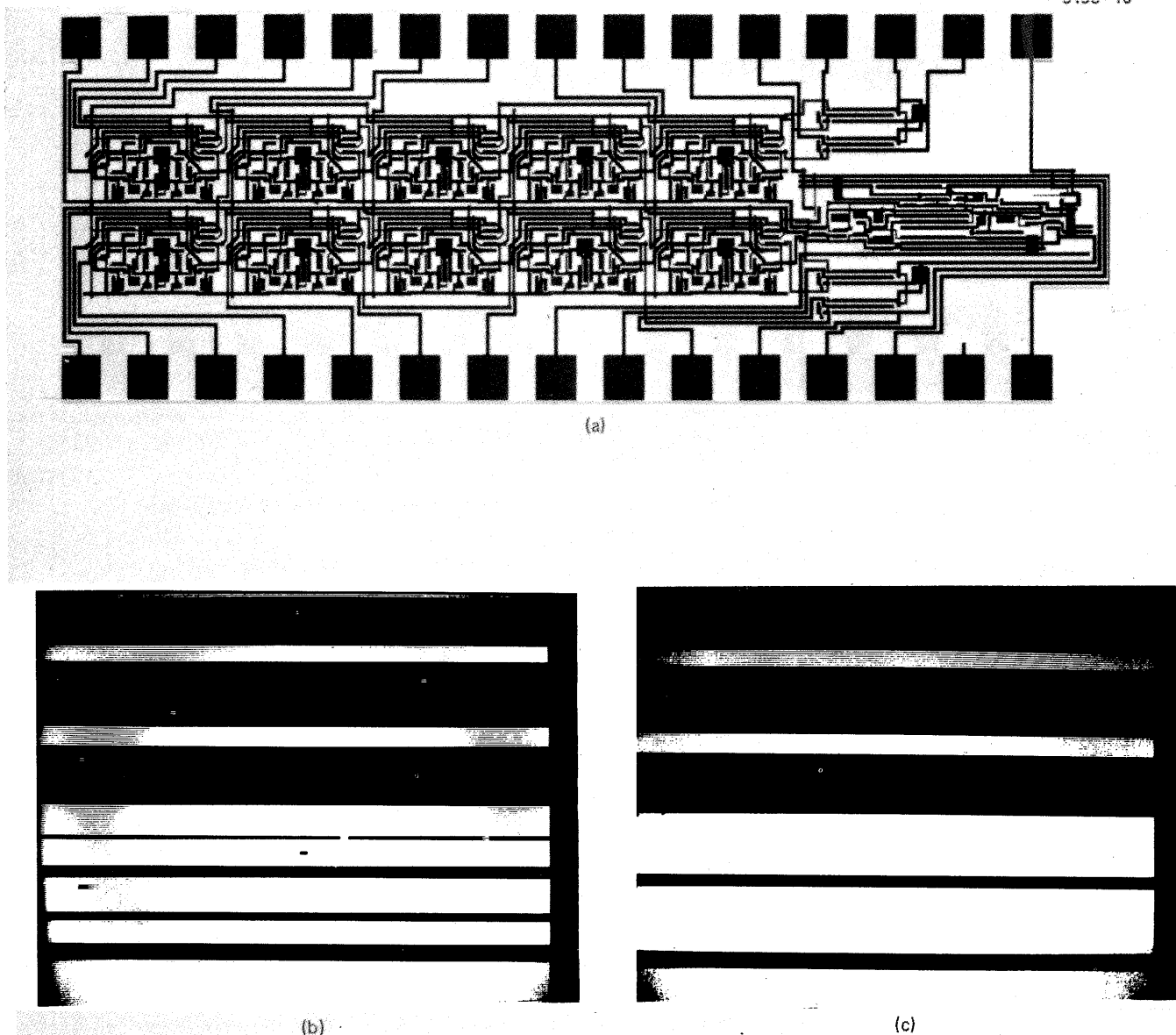


Figure 17—Median operator. (a) Circuit layout. (b) Input image. (c) Processed image.

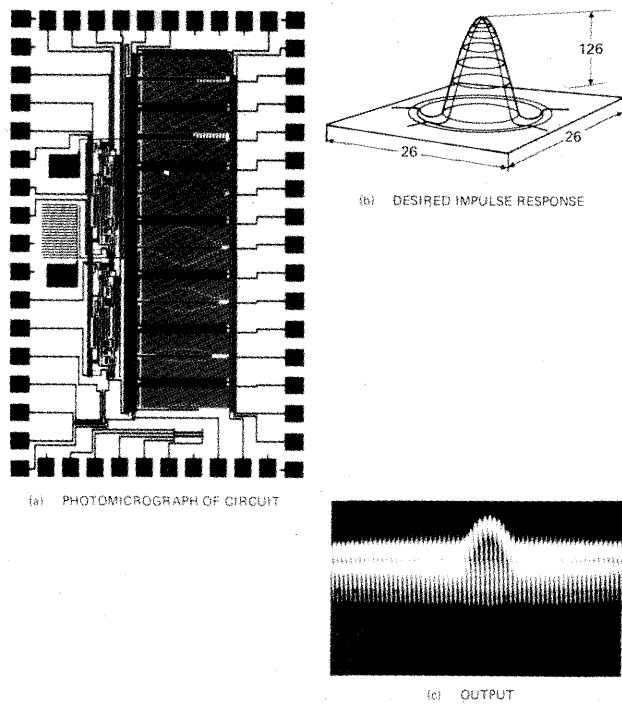


Figure 18—26×26 pixel bipolar convolution used for the primal sketch.

rected at the issues such as the optimum algorithm design for high density silicon, optimum partitioning between chips and means for providing an effective and fault tolerant control.

#### ACKNOWLEDGMENTS

I am indebted to my colleagues in the Exploratory Studies Department, Hughes Research Laboratories, Malibu, California for contributions to this paper and the Defense Advanced Research Project Agency (DARPA) Information Processing Techniques Office for supporting the work.

#### REFERENCES

1. Unger, S. H., "A Computer Oriented Toward Spatial Problems," *Proc. of the IRE*, Oct. '58, p. 1744.
2. McCormick, B. H., "The Illinois Pattern Recognition Computer—ILLIAC III," *IEEE Trans. Electronic Computers*, Dec. 1963, p. 791.
3. Gray, S. B., "Local Properties of Binary Images in Two Dimensions," *IEEE Trans. Computers*, v.c-20, no. 5, May 1971.
4. Preston, K., Jr., "Feature Extraction by Golay Hexagonal Pattern Transforms," *IEEE Trans. Computers*, v.c-20, no. 9, Sept. 1971.
5. Kruse, B., "A Parallel Picture Processing Machine," *IEEE Trans. Computers*, v.c-22, No. 12, Dec. 1973, p. 1075.
6. Duff, M. J. B., "CLIP 4: A Large Scale Integrated Circuit Array Parallel Processor," *Proc. of the Int'l Conf. On Patt. Recog.*, Coronado, Nov. 1976.
7. Mori, K., et al., "Design of Local Parallel Pattern Processor for Image Processing," *National Comp. Conf. Proc.*, 1978, p. 1025.
8. Barnes, G. H., et al., "The ILLIAC IV Computer," *IEEE Trans.*, c-17, v. 8, Aug. 68, p. 746.
9. Potter, J. L., "The STARAN Architecture and its Application to Image Processing and Pattern Recognition Algorithms," *Nat'l Comp. Conf. Proc.*, 1978, p. 1041.
10. Enslow, P. H., *Multiprocessors and Parallel Processing*, John Wiley and Sons, 1974, p. 139.
11. Sutherland, I. E., et al., "Microelectronics and Computer Science," *Scientific American*, Sept. 1977, vol. 237, No. 3, pp. 210-229.
12. Pratt, W. K., *Digital Image Processing*, John Wiley & Sons, N.Y., 1978.
13. Nevatia, R. et al., "Linear Feature Extraction," *Proc. ARPA Image Understanding Workshop*, Carnegie-Mellon, Nov. 1978, pp. 73-78.
14. Kuck, D. J., *The Structure of Computers and Computations*, John Wiley & Sons, N.Y., 1977.
15. Batcher, K. W., "Sorting Networks and Their Applications," *1968 Spring Joint Computer Conf., AFIPS Proc.*, Vol. 32, Washington, D.C. pp. 307-314, 1968.
16. Nudd, G. R., et al., "Application of Charge Coupled Device Technology in Two Dimensional Image Processing," 1978 International CCD Device and Application Conference, San Diego, Oct. 1978.
17. Nudd, G. R., et al., "A CCD Image Processor for Smart Sensor Applications," *Proceedings of Society of Photographic and Instrumentation Engineers Symposium*, San Diego, August 1978.
18. Marr, D., "Representing Visual Information," AIM-415, The Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Mass. 1977.
19. Dennis, J. B., et al., "A Computer Architecture for Highly Paralled Signal Processing," *Proc. of the Second Annual Symposium on Computer Architecture*, IEEE, pp. 126-132 (1975).
20. Misunas, D. P., "Structure Processing in a Data Flow Computer," *Proc. of the 1975 Sagamore Computer Conference on Parallel Computation*, IEEE, pp. 230-234, (August 1975).

# Map-guided interpretation of remotely-sensed imagery\*

by J. M. TENENBAUM, H. G. BARROW, R. C. BOLLES,  
M. A. FISCHLER, and H. C. WOLF

*SRI International*  
Menlo Park, California

## INTRODUCTION

Aerial and satellite imagery provide an economical means of gathering large amounts of data on the earth's resources and environment. However, except in the area of survey tasks such as crop inventories and land use that can be performed with multispectral analysis, there are few economically feasible techniques for automatically extracting the useful information from such imagery.

This paper describes some initial experiments in automating an important class of remote sensing tasks that involve continuous monitoring or tracking of predefined targets. *Monitoring* tasks are concerned with detecting anomalous conditions at specified geographic locations. Examples include monitoring particular industrial plants for thermal or chemical pollution, oil storage facilities for spillage, forests for fires, and reservoirs for water quality. *Tracking* is a variant of monitoring, concerned with determining the current geographic location of a slowly moving object or boundary whose position is known approximately from a previous determination. Examples include tracking icebergs, the spreading boundaries of a known oil spill, the perimeter of reservoirs (to assess changes in water volume), coastal shorelines (to assess erosion), and the width of rivers (to assess flood threat). For such tasks, an automated system is needed that can extract updated information as new imagery arrives and distribute it directly to interested users. Multispectral analysis, by itself, is inadequate because spatial structure and context are significant factors in interpretation.

A major problem in automating such tasks is locating the designated sites in sensed imagery, that may be taken from arbitrary viewpoints. Once the image locations of a site are known, many monitoring tasks are reduced to straightforward detection or classification problems. For example, once the precise pixel location of a river passing beside a manufacturing plant is known, pollution levels in the plant's effluents can, in principle, be determined by using conventional multispectral analysis. Similarly, forest fires can be detected by looking for infrared hot spots in known forested areas. Tracking slowly changing boundaries, such as the perimeters of water bodies, is also tremendously simplified

by knowledge of the boundaries' approximate prior location. Boundary detection and linking can then be accomplished using simple edge operators to verify precise edge locations along the predicted path.

To locate monitoring sites in an arbitrary image, we use a map in conjunction with an analytic camera model. The camera model is first calibrated in terms of known landmarks and then used to transform between map coordinates of designated sites and their corresponding image coordinates. By constraining where to look in an image and what to look for, a map and camera model greatly simplify the extraction of relevant information in complex aerial scenes.

## MAP-IMAGE CORRESPONDENCE

A fundamental requirement in exploiting a map is to establish the geometric correspondence between image and map coordinates, which then allows known ground sites to be located in the image. Ground locations have conventionally been determined by warping the current sensed image into correspondence with a reference image, based on a large number of local correlations [1]. The reference image serves as a map indicating locations in the sensed image that correspond to previously determined points of interest in the reference image. The process is computationally expensive and limited to cases where the reference and sensed images were obtained under similar viewing conditions.

To overcome these limitations, we abandon the use of a reference image and rely instead on a symbolic reference map containing explicit ground coordinates and elevations for all monitoring sites as well as landmarks (roads, coastlines, and so forth). The geometric correspondence between this map and the sensed image is established by calibrating an analytic camera model.

A typical camera model [2] has between five and seven parameters that specify focal length and the location and orientation of the camera (in map coordinates) when the image was taken. Once these parameters are known, the image coordinates corresponding to any map location can be determined precisely with straightforward trigonometry. (The camera location and map location jointly define a ray in space. The intersection of this ray with the image plane yields the desired image coordinates.) Since image coordi-

\* ©1979 IEEE. Reprinted from *Pattern Recognition and Image Processing* 1979 conference record.

nates are determined for the original unrectified image, expensive image warping is unnecessary.

#### *Map data base*

The map data base used in this research is essentially a compact three-dimensional description of the location and shapes of major landmarks and monitoring sites. Point features, such as road intersections, small buildings, and many

monitoring sites, are represented by their three-dimensional world coordinates and (where applicable) a list of characteristics to be monitored. Linear landmarks, such as roads and coastlines, are similarly represented as curve fragments with associated ordered lists of world coordinates. Ground coordinates are expressed in a standard reference frame, the UTM grid, with elevations expressed in meters above sea level. The data base can be accessed by location (e.g., What is at  $x, y, z$ ?), by entity name (e.g., What is the location of factory  $x$ ?), and by entity type (e.g., What factories are

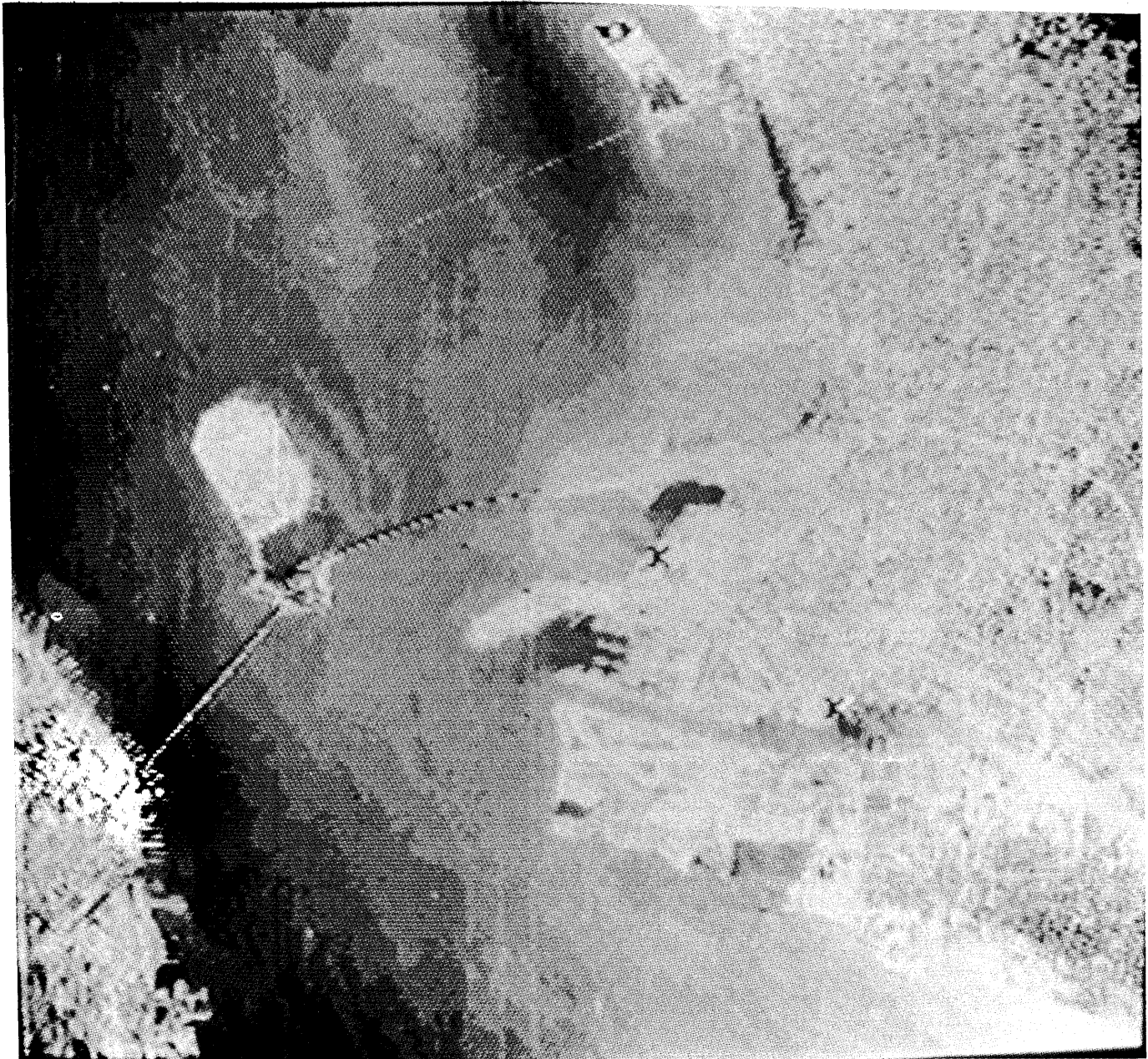


Figure 1—High altitude vertical mapping photograph of San Francisco Bay area.

there?). For further details on map representation, the reader is directed to Reference [3].

Our experimental domain throughout this project was the San Francisco Bay Area, as depicted in Figure 1. Figure 2 is a computer display of a simple map data base of this area. The map contains a major landmark (the coastline) and a number of representative monitoring sites, each designated by a cross. Longitude and latitude data for the on-line map were obtained interactively from a USGS map, using a digitizing table. Elevations were read off the map and entered manually via keyboard. Although displayed as a continuous trace, the coastline, in fact, is internally represented by just 100 discrete sample coordinates.

Several map data bases, each highlighting specific features (e.g., roads, railroad yards, piers) were used in experiments described in this report. These maps have not yet been integrated into a monolithic data base, although all software necessary to do so exists (Reference [3]).

#### Camera calibration

The traditional method of calibrating a camera model requires two stages: first, a number of known landmarks are independently located in the image; and second, the camera parameters are computed from the pairs of corresponding world and image locations, by solving an over-constrained set of equations [2,4].

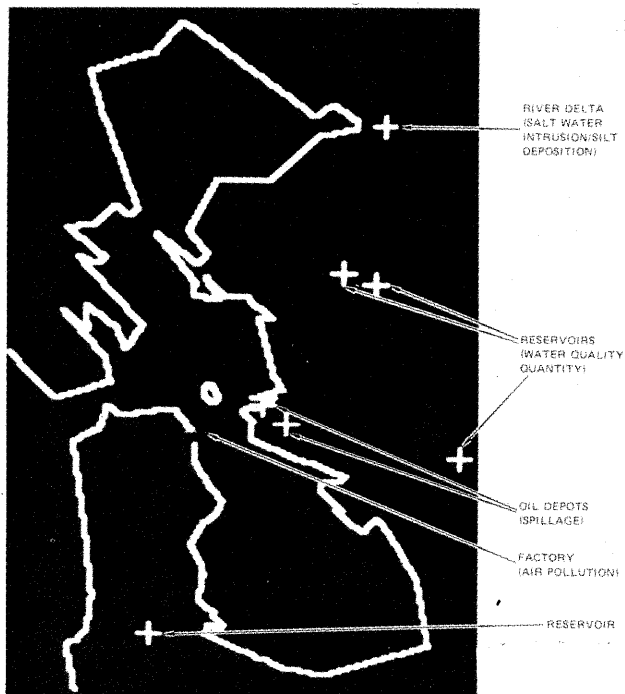


Figure 2—Computer display of a simple map database for the San Francisco Bay area, showing major landmark (coastline) and representative monitoring sites (crosses).

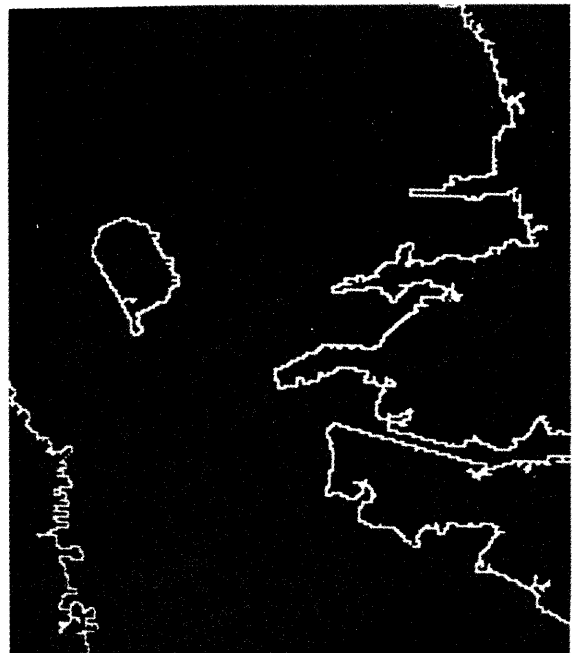


Figure 3—Coastline extracted by boundary follower.

The failings of the traditional method stem from the first stage: Landmarks are located in the sensed image by correlating with fragments of reference images. This requires reference images taken under the same viewing conditions as the current sensed image. Moreover, since landmarks are found individually, using only very local context (e.g., a small patch of surrounding image) and with no mutual constraints, false matches commonly occur. (The restriction to small features is mandated by the high cost of area correlation and by the fact that large image features correlate poorly over small changes in viewpoint.)

A new calibration procedure, called "Parametric Correspondence," was developed that overcomes these failings by integrating the landmark-matching and parameter solving steps and by using global shape rather than tonal appearance as the basis for matching. In this procedure, initial estimates of camera location and orientation are obtained on the basis of available navigational data. The camera model is then used to predict the appearance of landmarks in an image for this assumed viewpoint. Calibration is achieved by adjusting the camera parameters (i.e., the assumed viewpoint) until the predicted appearances of the landmarks optimally match a symbolic description extracted from the image.

A detailed description of parametric correspondence is given in Reference [5]. However, the essential ideas can be quickly grasped through an example. Figures 3-6 illustrate the process of establishing correspondence between the symbolic map of Figure 2 and the sensed image of Figure 1, using the coastline as a landmark.

First, a simple edge follower was used to trace the high contrast coastline in Figure 1, producing the edge image



Figure 4—Predicted image coordinates of coastline (based on navigational estimates of camera location and orientation) superimposed on extracted boundary.

shown in Figure 3. Next, using initial camera parameter values (estimated manually from navigational data provided with the image), the coastline coordinates in the map were transformed into corresponding image coordinates and overlaid on the extracted edge image (Figure 4). The average mean square distance between the extracted coastline and that predicted on the basis of the assumed viewpoint was seven pixels. A straightforward hill-climbing algorithm then adjusted the camera parameters to minimize this average

distance. Figure 5 shows the final state, in which the average distance has been reduced to 0.8 pixel.

Using the final parameter values, it is now possible to determine within a pixel the precise image locations corresponding to each monitoring site in the map. Only three sites are actually visible in this image: two oil depots and a coffee factory. These are shown in Figure 6, superimposed on the original image. The apparent misregistration in Figure 5 is actually the result of errors in contour extraction (Figure 3);

despite such errors, the global matching criteria is still able to achieve subpixel accuracy of the projected map points. Figures 7 and 8 provide two additional examples, illustrating the ability of the calibration process to place the map in Figure 2 into correspondence with imagery taken from arbitrary viewpoints.

Parametric correspondence has some significant advantages over conventional approaches to camera calibration

that depend on reference imagery. Computational requirements (both processing and memory) are sharply reduced because a symbolic map typically contains orders of magnitude less data than a reference image. Invariance to viewing conditions (viewpoint, spectral band, sun angle, etc.) is significantly improved because maps describe global shape characteristics that are relatively immune to seasonal and diurnal variation and to ambiguous matches. Moreover,

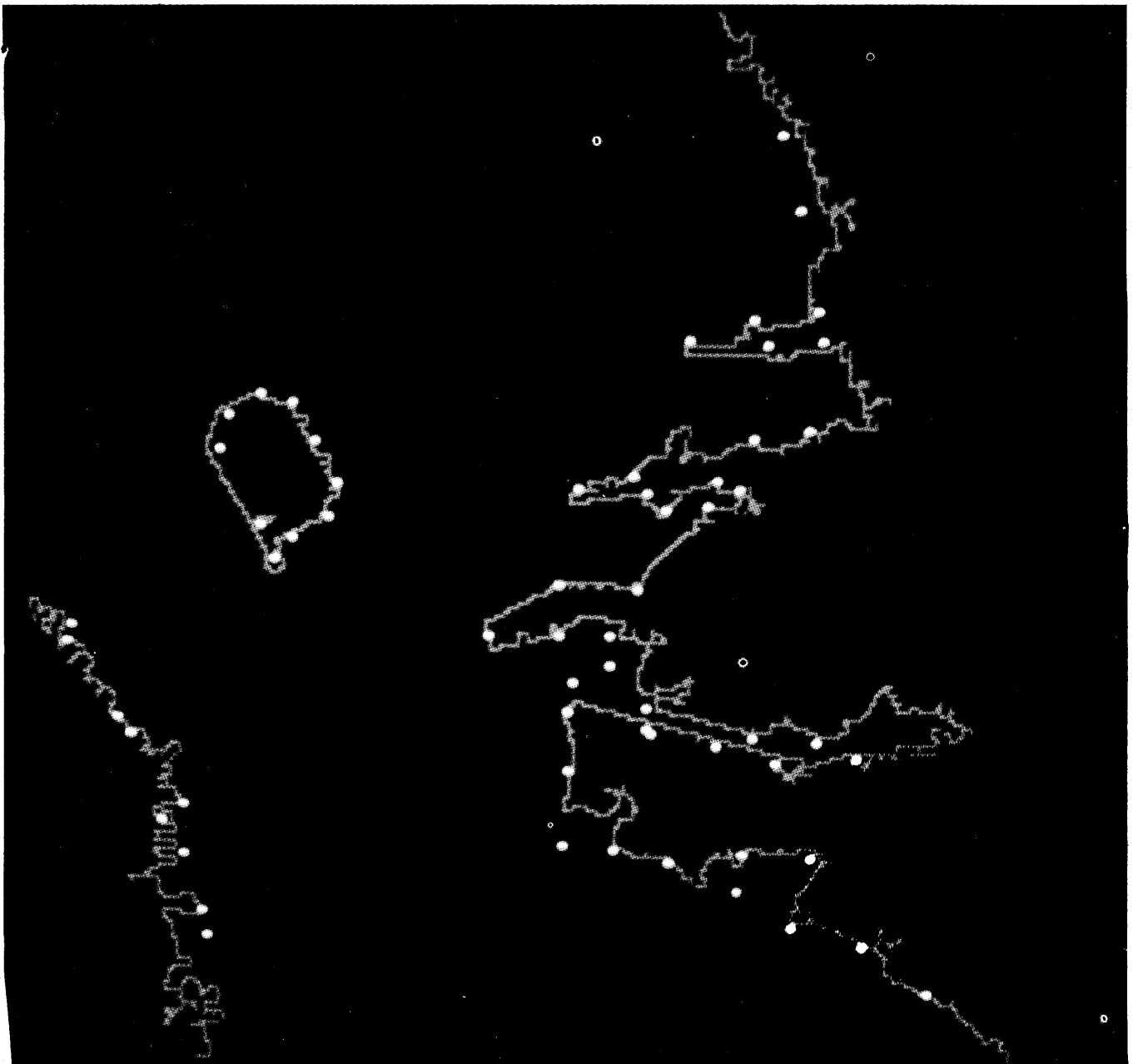


Figure 5—Predicted coastal coordinates after optimization of camera parameters.



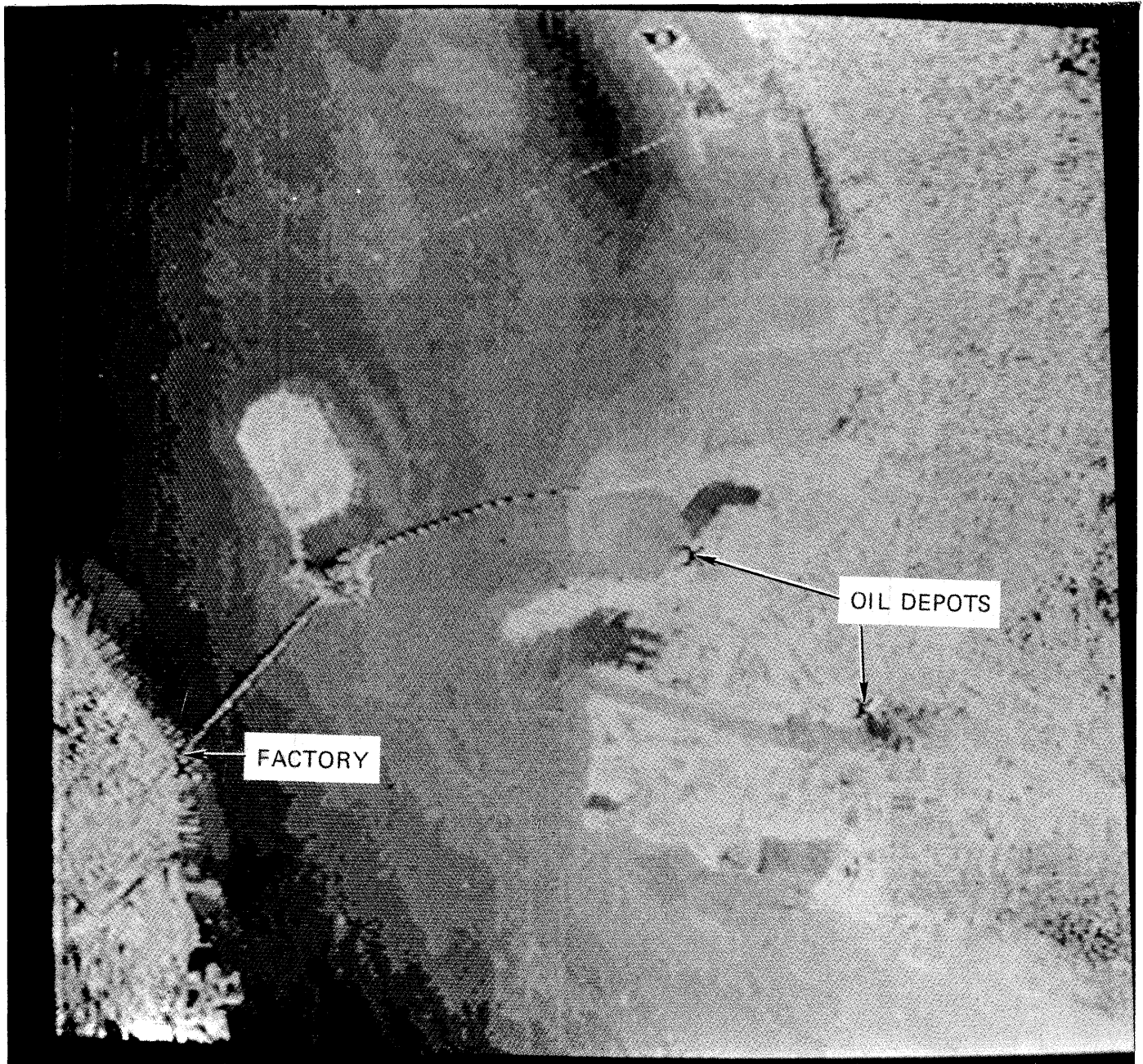


Figure 6—Predicted image locations of visible monitoring sites based on optimized parameters.

since shape information is projected through the camera model before matching, distortions due to viewpoint are no longer a problem. A detailed discussion of these advantages appears in Reference [5].

#### MAP-GUIDED MONITORING

Having placed an image into parametric correspondence with a three-dimensional map, it is possible to predict the image coordinates of any feature in the map and, conversely,

to predict the map features corresponding to any point in the image. Given this capability, many basic monitoring tasks of the type discussed in the introduction can be automated using straightforward image-analysis techniques. In Figure 8, for example, one could, in principle, test the pixels located in reservoirs for water quality, the pixels located in shipping channels beside oil depots for evidence of spillage, the pixel located at the industrial plant for evidence of particulates, and the pixel located at the Sacramento River Delta for evidence of salt water intrusion.

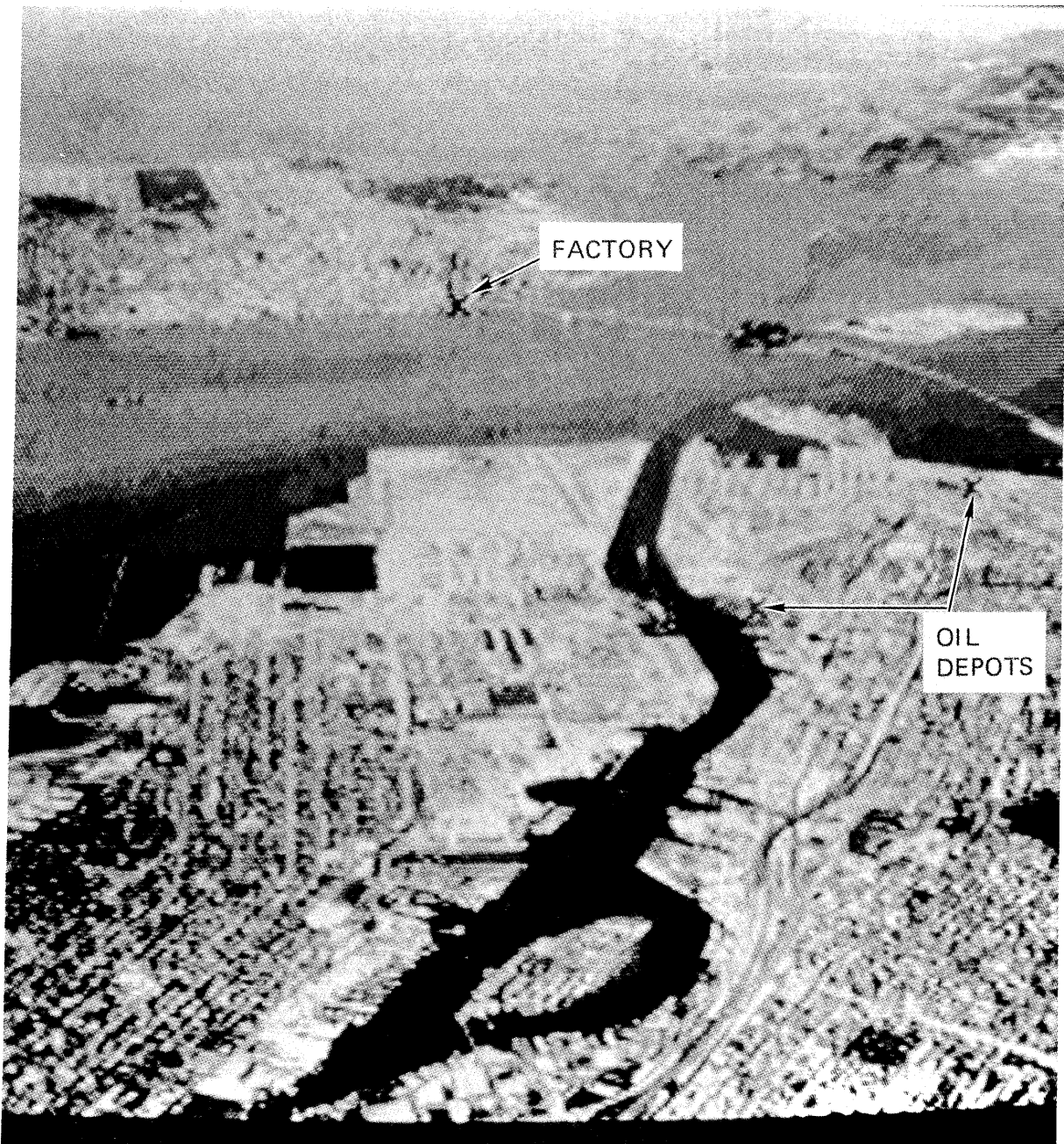


Figure 7—Predicted locations of visible monitoring sites in an oblique view looking west from Alameda.

These examples fall within the competence of traditional multispectral analysis programs which uniformly process all pixels in an image and produce a statistical result. For such tasks, the primary advantages of map guidance are an enormous reduction in the number of pixels to be processed, potentially enhanced discrimination (resulting from the ability to optimize classification criteria at each site), and geographically specific results that are generally more useful than statistical summaries. In more complex interpretation tasks, where spatial structure and context are important, the

benefits of map guidance are more profound. Four representative experiments will now be described.

#### *Reservoir monitoring*

Consider first the problem of monitoring the water level of a reservoir. Water level, of course, is not directly measurable from an aerial image; some additional information or

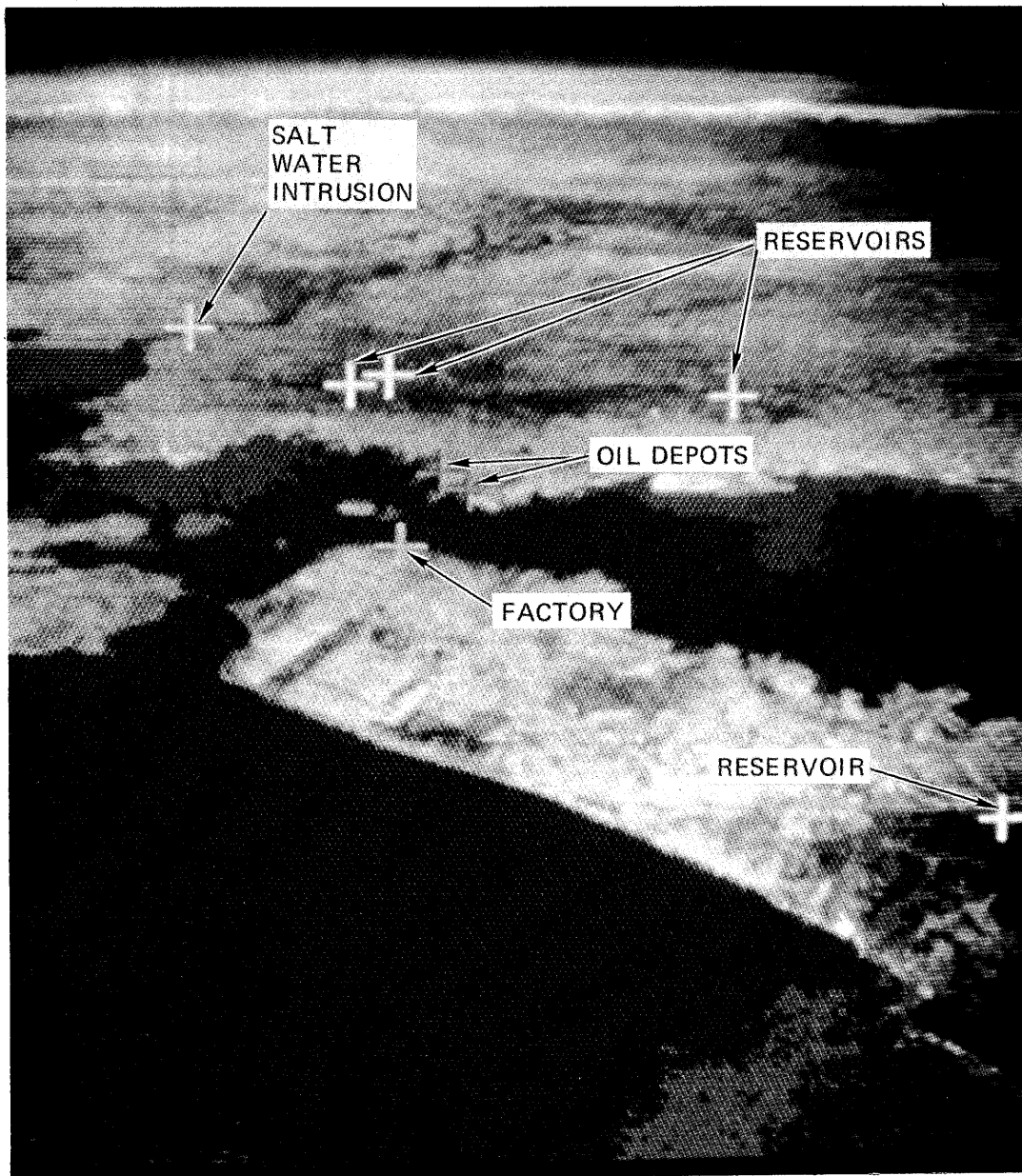


Figure 8—Predicted locations of visible monitoring sites in a high altitude oblique view looking east from the Pacific Ocean.

constraint is needed. The required information can be obtained from a terrain map in registration with the image.

As the water level rises and falls, the outline of the reservoir expands and contracts in a predictable way to follow the elevation contours of the terrain (see Figure 9). Thus water level can be determined by extracting the outline of the reservoir in the image and determining its location with respect to known elevation contours. Knowing the water level, one can then integrate over the corresponding region of flooded terrain to determine the volume of stored water.

(The function relating water volume and water level is monotonic and can be tabulated for each reservoir.)

Since the surface of a reservoir is flat, the water level can be determined without a complete outline; the image coordinates of even a single point on the reservoir boundary would, in principle, suffice. In practice, elevations can be determined for a number of boundary points and averaged together to compensate for statistical uncertainties in estimating the precise image coordinates of each boundary point. (Concentrating the boundary samples where terrain

slope is most gradual maximizes the sensitivity of edge location to changes in water level. See Figure 9(b).) The resulting distribution of elevations, which should be tightly clustered, provides a check on the quality of the map-image correspondence.

A reservoir monitoring procedure incorporating these ideas was implemented. First, geometric correspondence was established between the sensed image and a contour map of the terrain using the techniques described in the previous section. Correspondence was based on geographically stable landmarks unrelated to reservoir boundaries.

Second, the image coordinates of selected points on the reservoir boundary were determined to subpixel precision by analyzing the gradient of intensity along a line in the image perpendicular to the elevation contours at each point. The analysis was restricted to a contour interval bracketing the water level observed in a previously analyzed image. This constraint not only reduced computation but also served as an effective contextual filter for discriminating irrelevant intensity discontinuities, arising, for example, from other nearby bodies of water.

Third, the water level corresponding to each detected boundary point was obtained by linearly interpolating the elevations of the terrain contours used to delimit boundary detection.

Finally, the water volume corresponding to the average water level was obtained by table lookup.

Steps (2)-(4) are repeated for each reservoir in an image containing more than one.

The above procedure was tested on a set of images of Briones reservoir, the rightmost of the twin reservoirs in the upper center of Figure 8. Figure 10 is a higher resolution

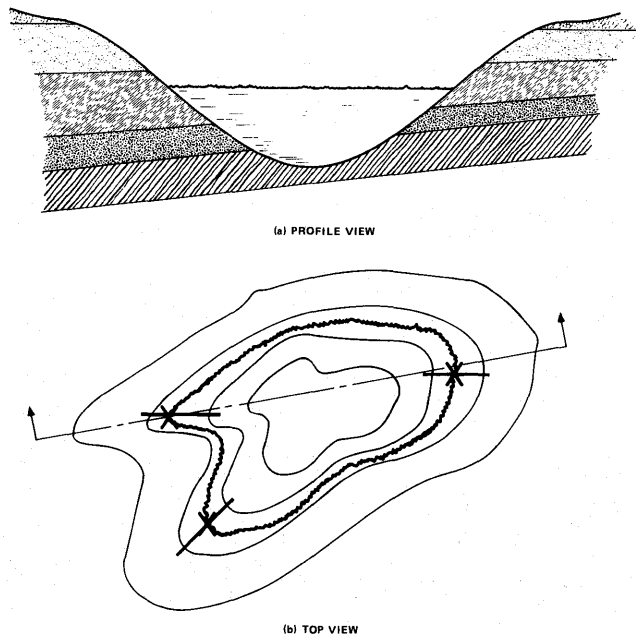


Figure 9—Relationship of water level to topography of terrain.

image of the Briones shoreline with elevation contours superimposed. The lines in Figure 11 indicate selected perpendiculars between the 500 and 550 elevation contours where the terrain slope is most gradual. The location of the land/water boundary along each of these lines was assigned to the point of maximal intensity discontinuity, as shown in Figure 12.

The water level corresponding to each boundary point was computed by interpolation. The mean water level in the present image of Briones, based on interpolating 170 boundary points, was determined to be 523.8 feet. This is within a foot of the ground-truth figure provided by the reservoir operator and corresponds to about a one percent error in volume. The accuracy of this approach is limited by the accuracy of the terrain map, the quality of map-image correspondence, and the precision with which the land/water interface can be located in an image. These factors are discussed further in Reference [6].

Reservoir monitoring is an instance of a generic class of tasks in which it is necessary to determine the precise path through an image of a linear feature (e.g., shoreline, river, road) whose location and shape are known, perhaps only approximately, from a map. Maps can be used in such tasks to facilitate both the process of locating the boundary in the image and the subsequent interpretation of boundary characteristics in terms meaningful to an application (e.g., interpreting image coordinates as water levels). Applications of map-guided boundary verification might include monitoring river widths (and heights) for flood threat, monitoring coastlines for erosion, and monitoring river deltas for excessive silt deposit. Unlike reservoir monitoring, extensive manual ground-based monitoring is not economically feasible in these applications.

### Road monitoring

Locating known roads in an aerial image is a prerequisite for a variety of applications ranging from vehicle monitoring [7] to map updating. Finding roads is somewhat different from finding reservoir boundaries in that a thin linear feature is involved and a continuous path is needed.

Conventional sequential line-tracking algorithms are unsuitable because they are easily sidetracked whenever either the local evidence for a line is weak or other lines are present in close proximity. These contingencies arise frequently in aerial imagery because roads are usually clustered into networks and pass regularly through heavily textured areas where one or even both edges may be locally obscured.

To overcome these problems, a line-tracing algorithm was developed that uses a rough prediction of the path of a road, provided by a map, as a guide in determining the precise path. The map information constrains the analysis to relevant parts of the image and is used to bridge gaps where local pictorial evidence is weak or ambiguous. The algorithm operates by applying specially developed line and edge detectors in the vicinity of the predicted road path and then

uses a parallel dynamic programming algorithm to find a globally optimal path through the local feature values. Further technical details can be found in Reference [8].

Figures 13-16 show the tracing algorithm in action. Figure 13 is an aerial image of a rural area taken for a U. S. Geological Survey mapping project. The portion shown has been digitized into  $256 \times 256$  pixels (representing 20-foot squares on the ground), each having one of 256 brightness levels. Overlaid on the image is a road path predicted from a map

with standard (50-foot) cartographic accuracy. A local line detector was applied at all image points within a band centered on this guideline. The system then found the lowest-cost path from the start of the guideline to the finish, where the incremental path cost between adjacent image points was an inverse function of the local line detector score. The path so traced is displayed in Figure 14. Figure 15 shows the results of tracing many of the roads visible in the image. Note that the program has traced the center line of the wide road

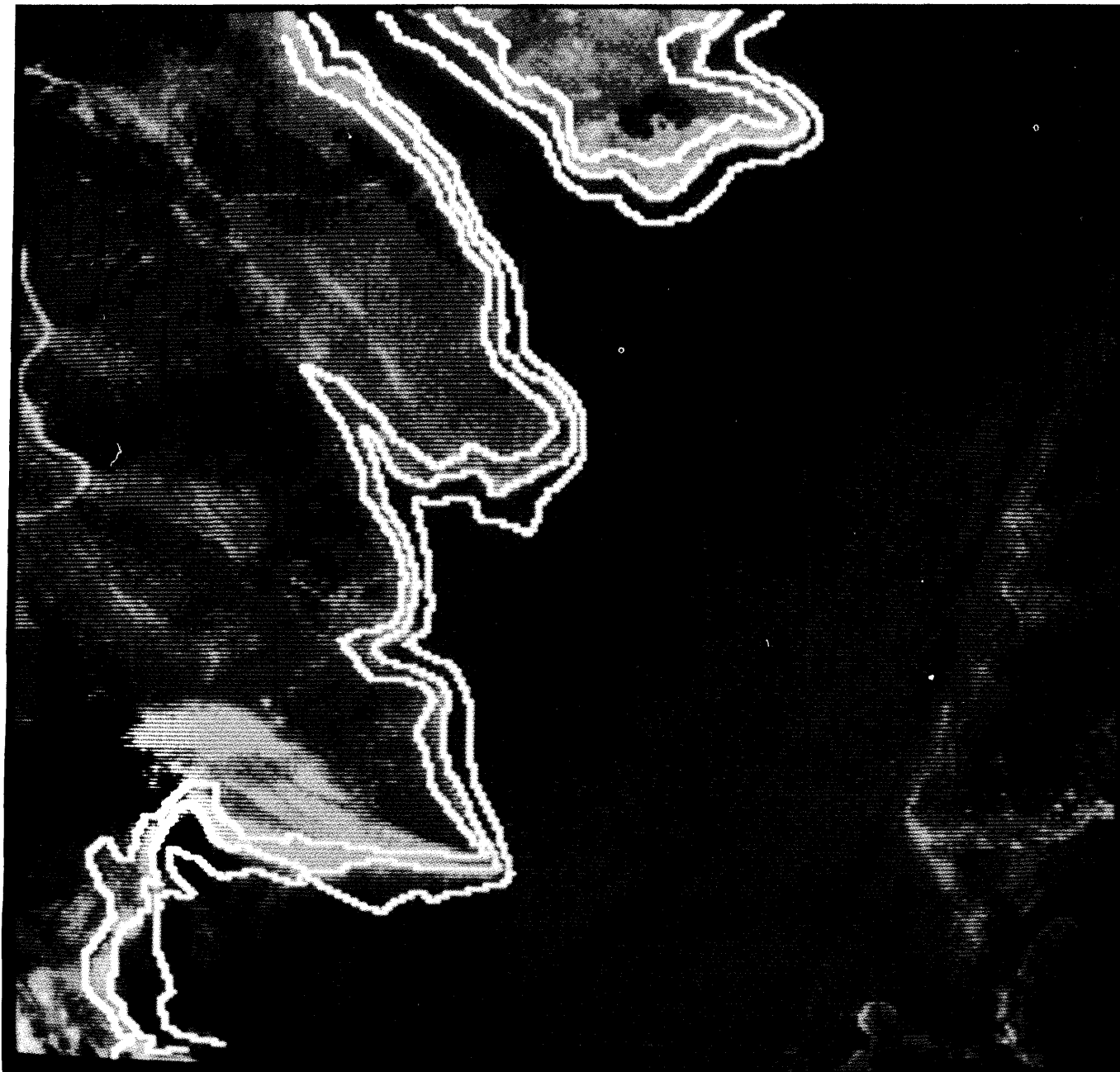


Figure 10—Terrain contours superimposed on image of Briones Reservoir.

and that it has performed extremely well in areas in which the road is faint or partially obscured, such as at the lower left and the upper right of the image. Figure 16 shows the results of guided road tracing in an urban area containing many intersecting streets. The tracings have been fitted with straight line segments to cartographic accuracy. The results here, too, are extremely good.

Although we have performed only a limited number of experiments with guided tracing, the results have been most

encouraging. The system is capable of tracing linear features that are hard even for a human to discern through a wide range of terrain types and environments. It needs relatively little guidance; but the more guidance it is given, the more reliable and efficient is its performance. It can accept guidance interactively (via light pen), as well as from preexisting maps. Interactive guidance is useful in map updating, allowing new roads to be carefully traced on the basis of a quick, light pen sketch.

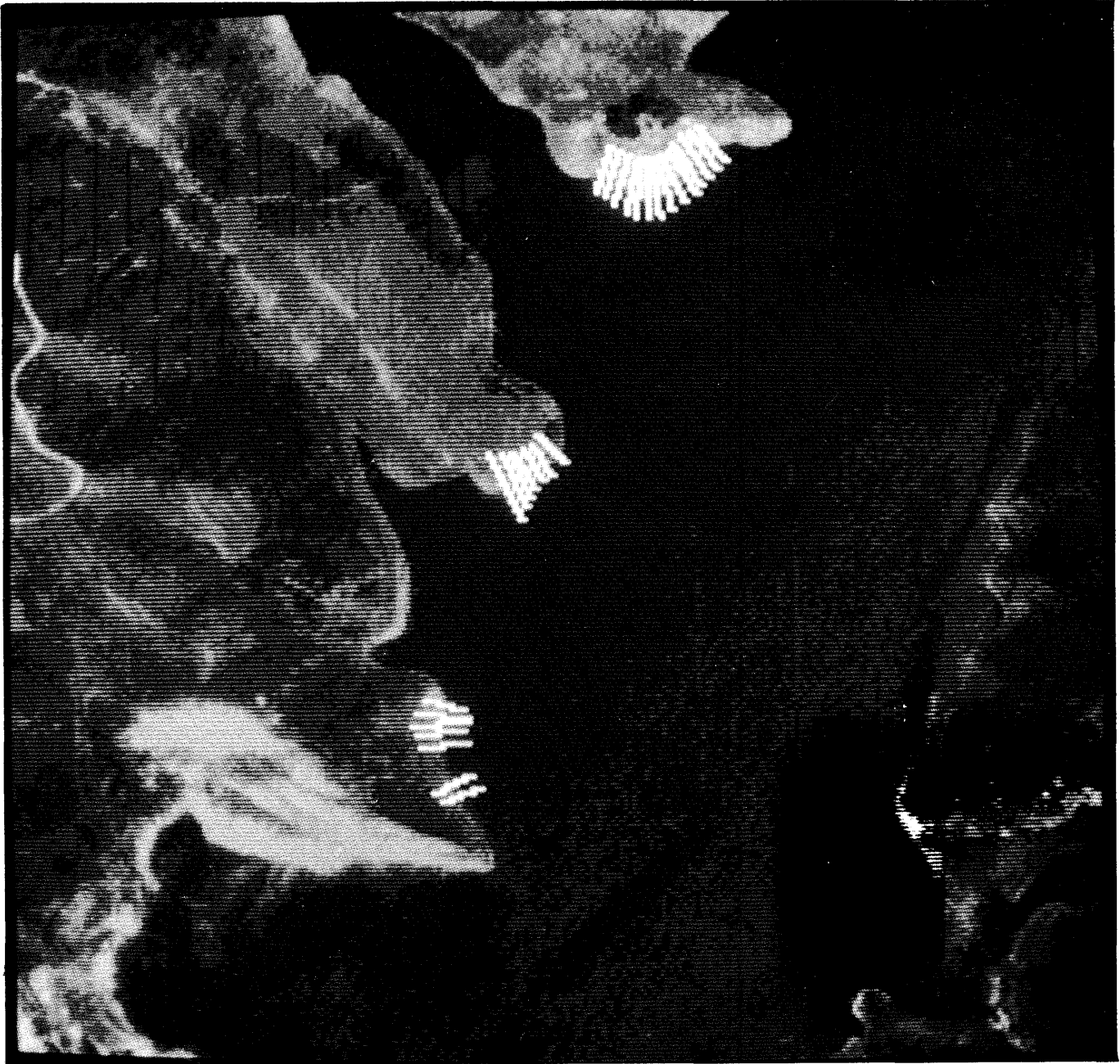


Figure 11—Lines designating location for determination of land-water boundary.

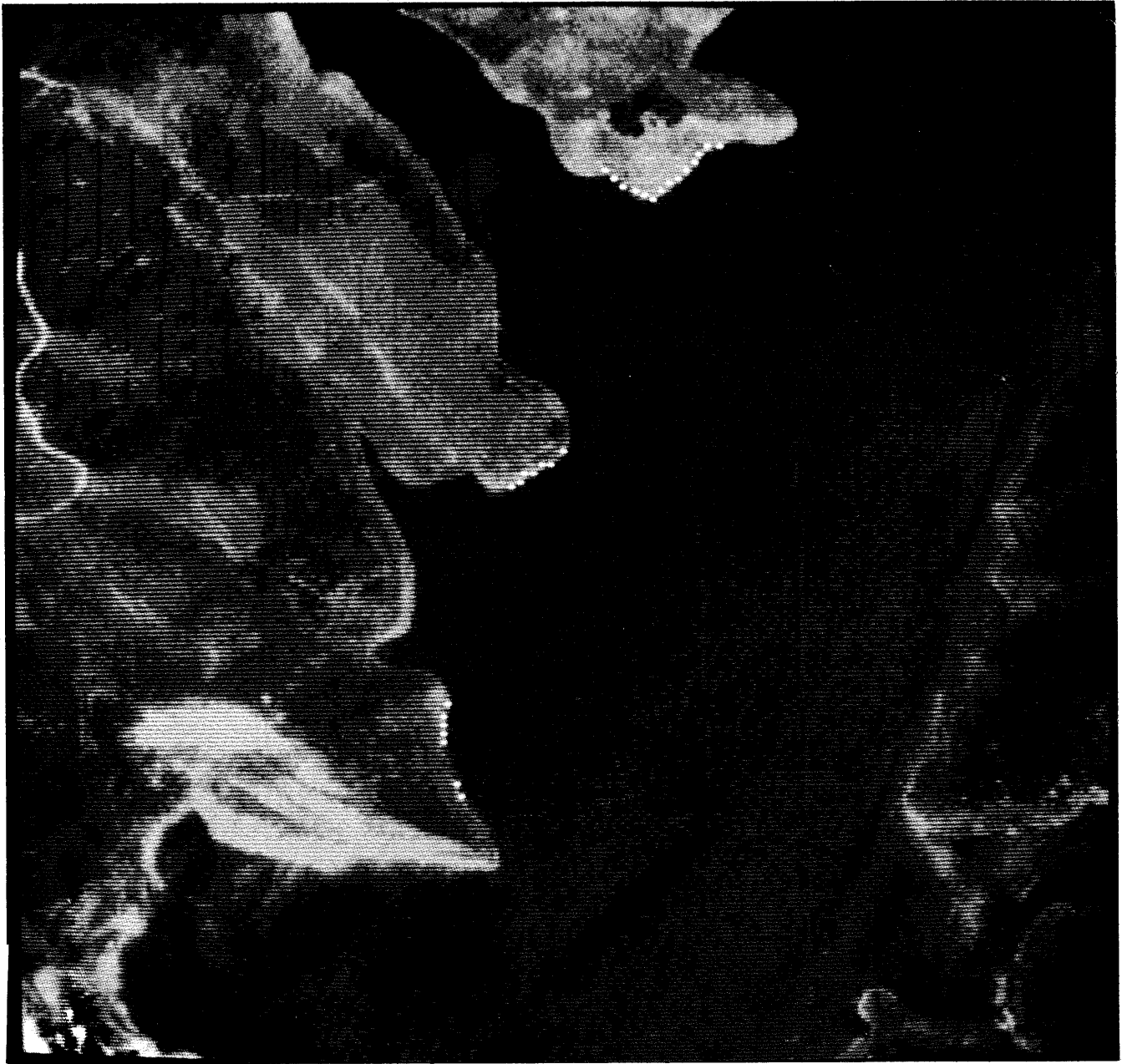


Figure 12—Locations of land-water boundary assigned to points of highest local gradient along lines shown in Figure 11.

Map-guided tracing linear features is a requirement that arises in a variety of remote sensing tasks, for example, in the monitoring of rivers and railroad lines. Given suitable operators for detecting local evidence, the optimal path algorithm used to obtain a continuous road track should also work equally well in these other line tracing applications.

#### *Object verification tasks*

Railroad and highway monitoring are two examples of a generic class of remote sensing applications we shall call object verification tasks. Such tasks entail the detection, mensuration, or counting of specified entities whose possible

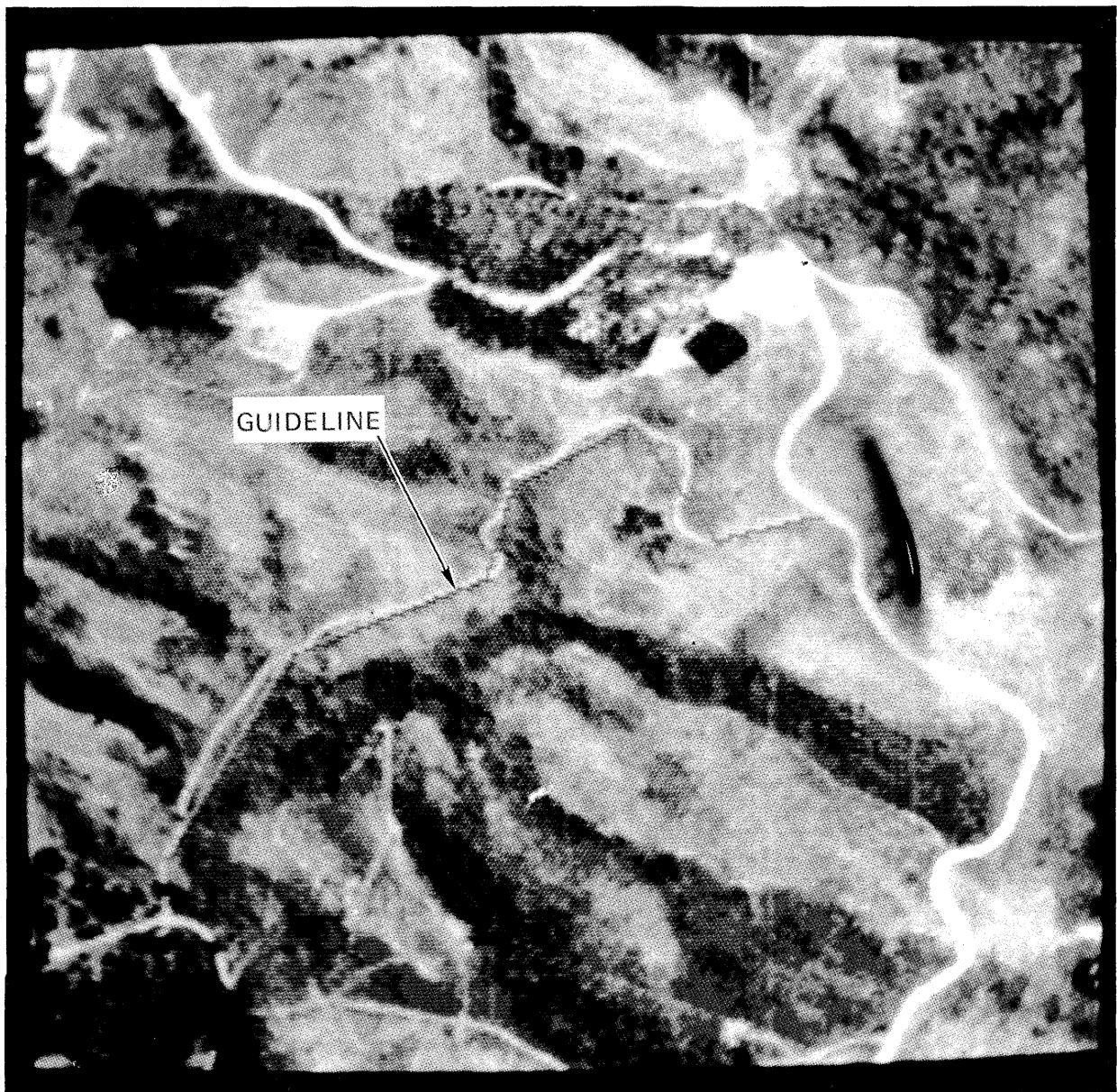


Figure 13—A rural road with guideline.

locations and orientations in the image can be constrained by a map. The general approach is to determine the image coordinates for a reference structure (such as a railroad track, ship berth, or road) and then apply special-purpose operators to detect objects of interest (such as boxcars, ships, or cars). For example, we have implemented a boxcar-counting routine that analyzes the intensity profiles along predicted paths of railroad track in an image, looking for possible ends of trains and gaps between cars. Such events usually appear as step changes in brightness and dark, transverse lines, respectively. Hypothesized gaps and ends are

interpreted in the context of knowledge about trains (e.g., standard car lengths and allowed intercar gap widths) and about the characteristics of empty track to prune artifacts and improve the overall reliability of interpretation. The program then reports the number of cars classified by length [8]. We have also implemented a ship-monitoring program that analyzes intensity patterns alongside predicted berth locations in a harbor to distinguish ships from water. (Water characteristically has a low density of edges, [9].) Railroad monitoring is illustrated in Figure 17 and ship monitoring in Figure 18.



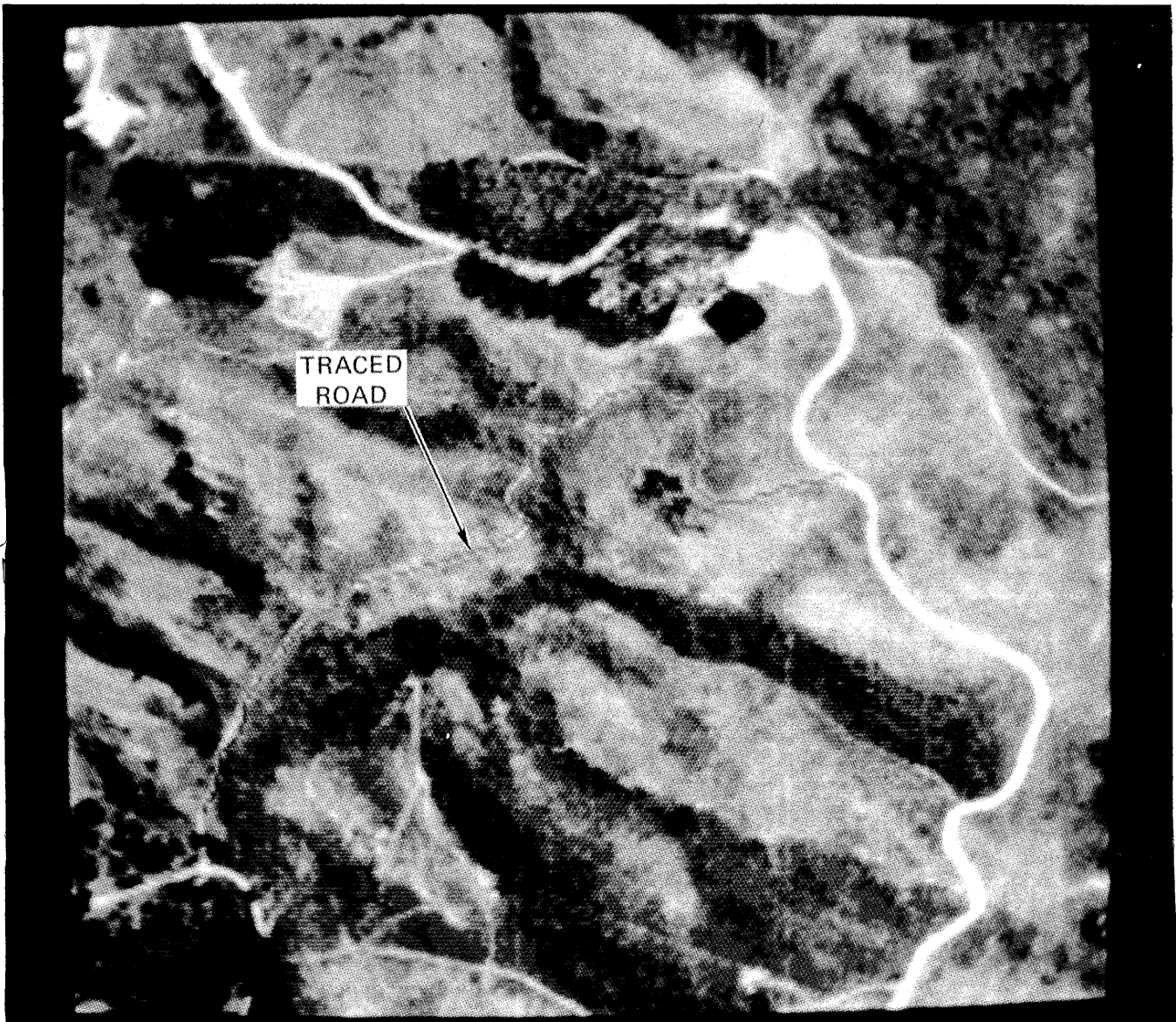


Figure 14—Output of guided tracing algorithm.

The key to automating both tasks lies in using a map to define a highly constrained context (i.e., area of the image) in which relatively simple tests can be used to distinguish objects of interest. Knowing the locations of tracks, for example, reduces the task of boxcar counting to a one-dimensional, template-matching problem, while knowing the locations of berths reduces ship finding to a trivial discrimination task. We believe that boxcar counting and ship monitoring are representative of a broad class of object-verification tasks that includes counting planes on runways and cars on highways, for which similar monitoring programs can be developed.

#### CONCLUDING COMMENTS

This paper has described a map-guided approach for automating an important class of remote sensing tasks involving long-term monitoring of predefined ground sites. The key idea is the use of a map in conjunction with an analytic camera model to constrain where to look in an image and what to look for. With map-guidance, many previously intractable monitoring tasks become feasible, in some cases even easy, to automate.

The map-guided approach has some potentially significant advantages over the exhaustive statistical style of processing

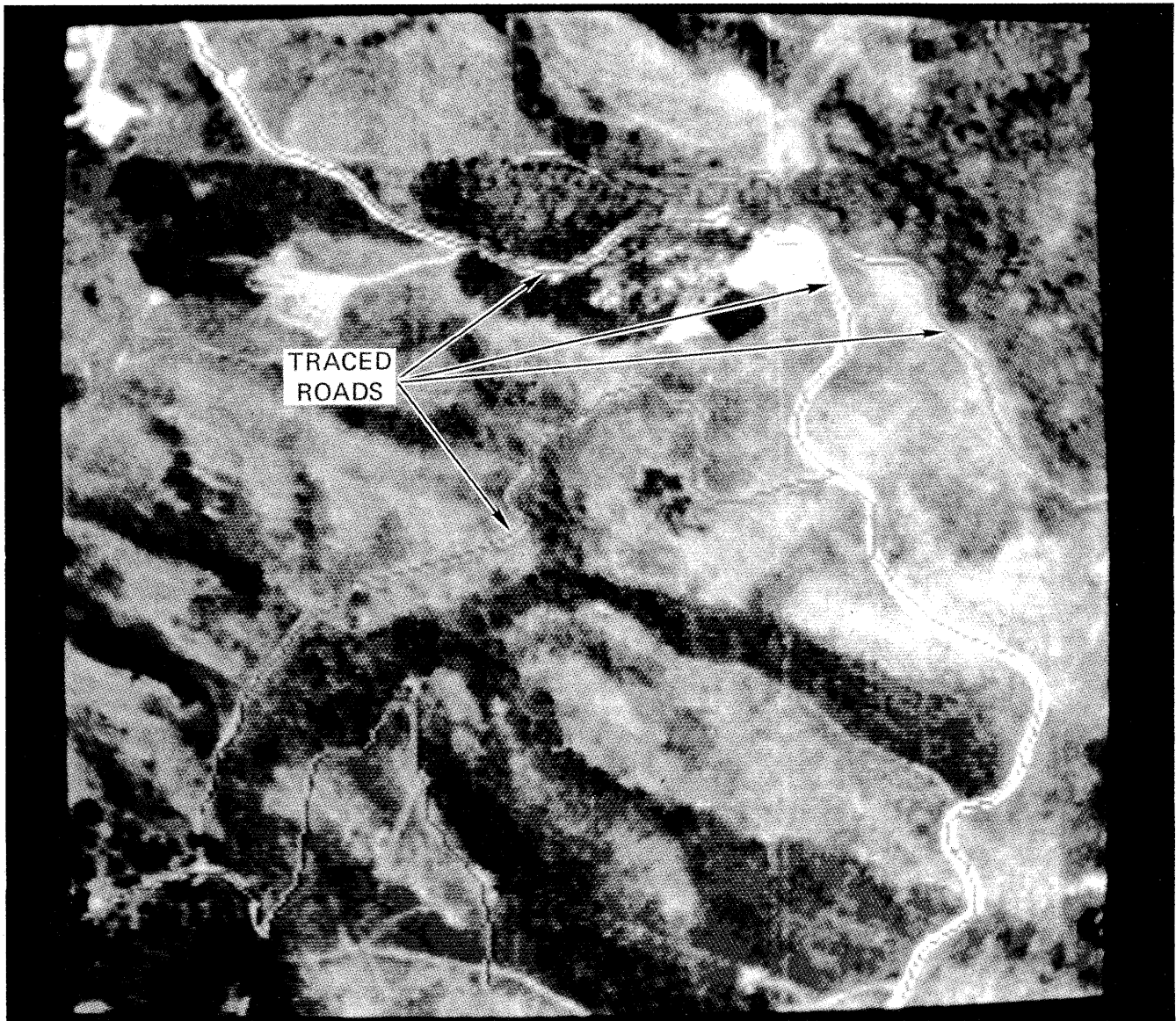


Figure 15—Guided tracing of several rural roads.

currently used in applications such as crop classification. First, processing can be focused on the relevant portions of an image, sharply reducing computational costs and making feasible the use of sophisticated forms of analysis (involving texture, spatial patterns, and the like) that would be utterly impractical to apply at each pixel (16 million in a typical  $4000 \times 4000$  LANDSAT image). Second, analysis routines can be simplified and made more reliable by exploiting knowledge of what to look for at each site. For example, classification criteria can be optimally tuned to discriminate the few relevant alternatives at each location. Finally, a map-guided analysis yields geographically specific results that are much more useful than conventional statistical summaries:

Knowing that a particular factory is emitting excessive  $\text{SO}_2$  is much more useful, for example, than knowing that 1 percent of 16 million pixels are polluted.

The practicality of automating monitoring tasks using the approach we have described depends, of course, on the availability of high resolution satellite imagery and satellite sensors that can be modeled analytically. Assuming these are forthcoming, the payoffs from automated monitoring could be substantial. We envisage systems that would extract updated information automatically as new imagery arrived and distribute it to interested users on a subscription basis. Initially, the analysis could be performed at existing ground-based data-processing facilities with only modest in-

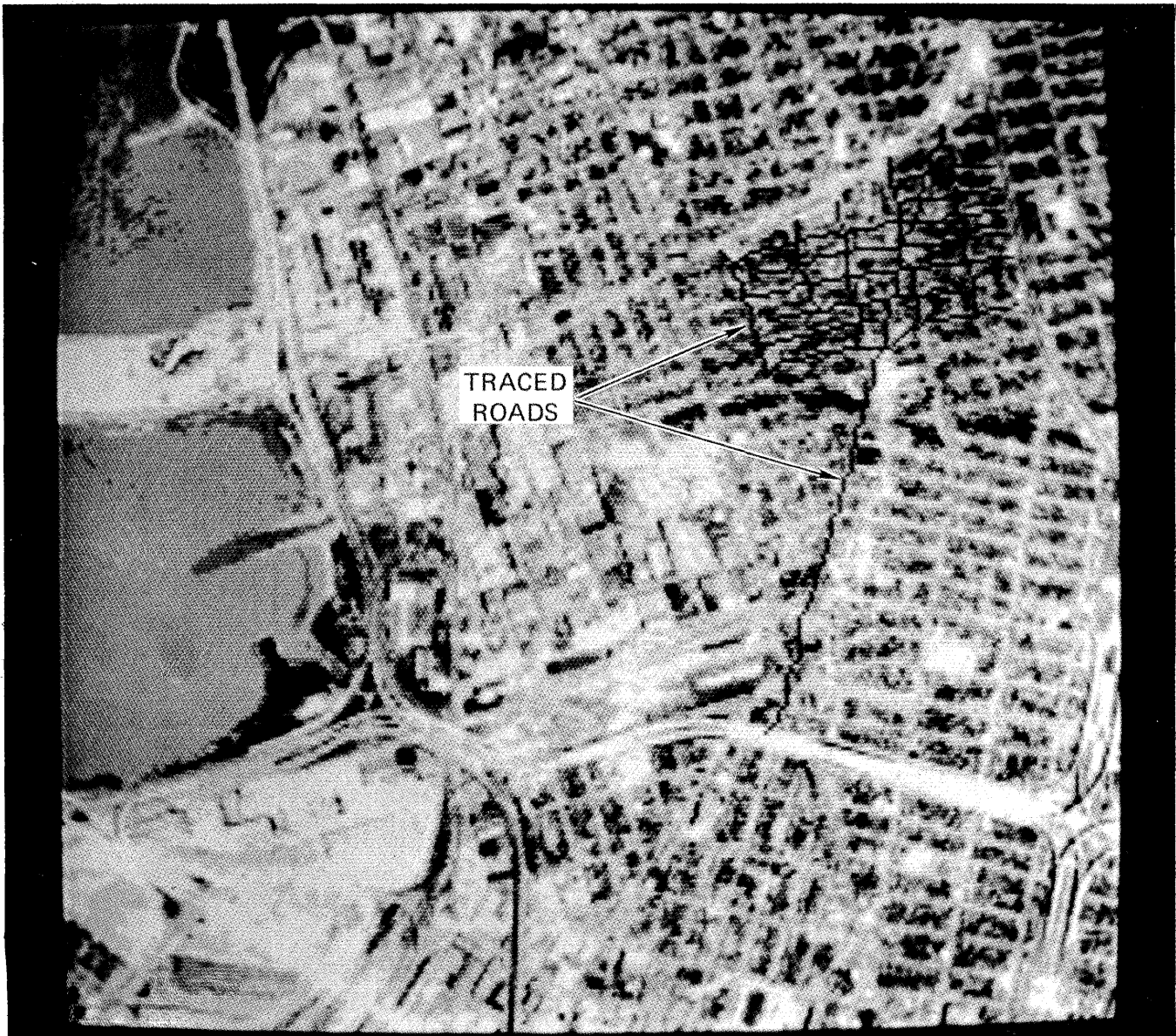


Figure 16—Guided tracing of several urban streets.

creases in computational load. Ultimately, the information could be extracted on-board satellites dedicated to specific monitoring functions and relayed direct to users via communication satellites. On-board processing appears feasible because of the dramatic reductions in computation made possible by the concept of map-guided image analysis.

For routine monitoring tasks with large user constituencies, centralized information extraction should significantly reduce the overheads of storing, retrieving, and distributing

large volumes of data. Moreover, it would eliminate the need for installing image analysis facilities at many user sites.

#### ACKNOWLEDGMENTS

This research was supported by the National Aeronautics and Space Administration under Contract No. NASW-2865 and by the Advanced Research Projects Agency under Contract No. DAAG29-76-C-0057.

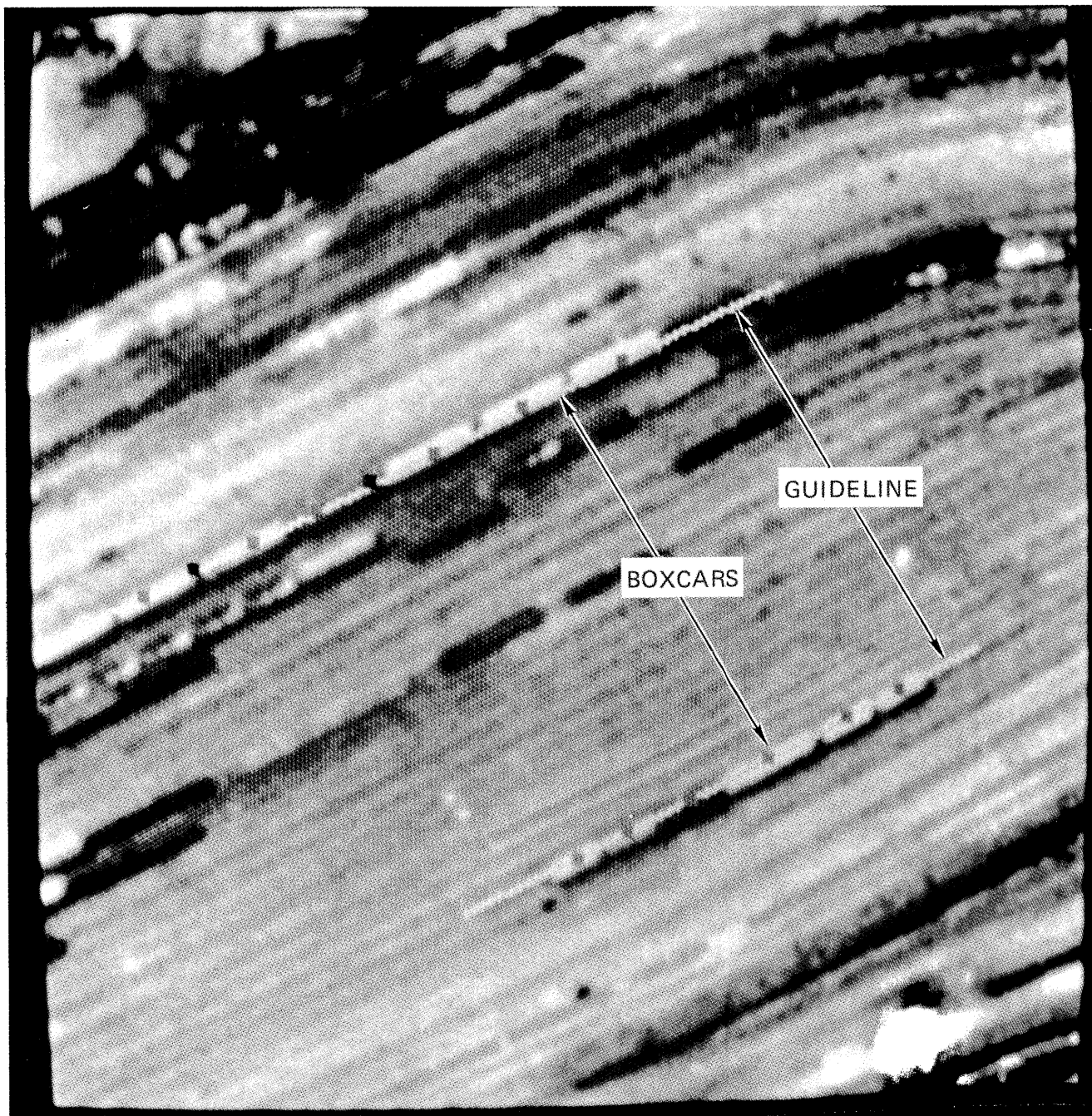


Figure 17—Automated boxcar counting. Lines indicating track locations were traced interactively in this example but could have been obtained by putting in correspondence with a three-dimensional map of the railyard, as in the ship example of Figure 18. Statistical operators are flown along tracks to detect

dark transverse lines that are characteristic of gaps between boxcars. Boxcars are indicated by dots whenever the spacing between hypothesized gaps is inconsistent with knowledge of standard car lengths.

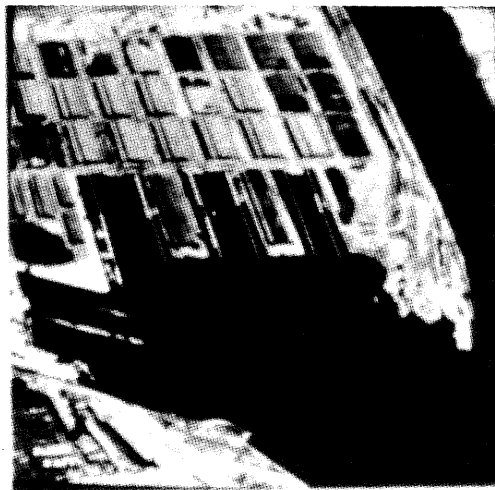
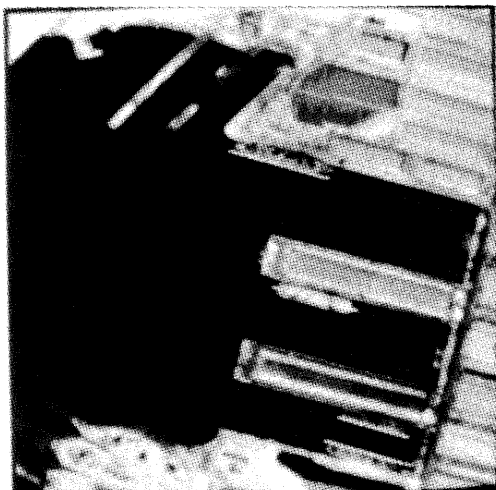


Figure 18—Automatic ship monitoring. The guidelines indicating known berth locations were obtained for both images from the same three-dimensional map of Oakland Harbor, based on determination of viewpoint for each image. The

light, wiggly lines beside the berths indicate regions of high edge content, characteristic of ships.

## REFERENCES

1. Bernstein, R., "Digital Image Processing of Earth Observation Sensor Data," *IBM Journal of Research and Development*, Vol. 20, No. 1 (January 1976).
2. Sobel, I., "On Calibrating Computer Controlled Cameras for Perceiving 3-D Scenes," *Artificial Intelligence*, Vol. 5, pp. 185-198 (1974).
3. Barrow, H. G., "Interactive Aids for Cartography and Photo Interpretation," Semiannual Technical Report (Appendix A), Contract DAAG29-76-C-0057, SRI Project 5300, SRI International, Menlo Park, California (December 1977).
4. Duda, R., and Hart, P., *Pattern Classification and Scene Analysis* (John Wiley & Sons, Inc., New York, 1973).
5. Barrow, H. G., et al., "Parametric Correspondence and Chamfer Matching: Two New Techniques for Image Matching," in *Proc. Fifth Intl. Joint Conference on Artificial Intelligence* (Cambridge, Massachusetts, August 1977).
6. Tenenbaum, J. M., Fischler, M. A. and Wolf, H. C., "A Scene Analysis Approach to Remote Sensing," SRI A.I. Center Technical Note 173, SRI International, Menlo Park, California (October, 1978).
7. Barrow, H. G., "Interactive Aids for Cartography and Photo Interpretation," Semiannual Technical Report, Contract DAAG29-76-C-0057, SRI Project 5300, SRI International, Menlo Park, California (June 1978).
8. Barrow, H. G., "Interactive Aids for Cartography and Photo Interpretation," Semiannual Technical Report, Contract DAAG29-76-C;0057, SRI Project 5300, Stanford Research Institute, Menlo Park, California (November 1976).
9. Barrow, H. G., "Interactive Aids for Cartography and Photo Interpretation," Semiannual Technical Report, Contract DAAG29-76-C-0057, SRI Project 5300, Stanford Research Institute, Menlo Park, California (May 1977).

# CCITT standardization for digital facsimile

by T. L. McCULLOUGH

3M Company  
St. Paul, Minnesota

## INTRODUCTION

In November, 1979, the CCITT Study Group XIV met to complete, among other items, several years of work on the Recommendations defining Group 3 digital facsimile. This paper reviews the results of this work in terms of technical implication, terminal features, and facsimile services. In particular, CCITT Recommendations T.4 and T.30 are discussed. (These Recommendations will be finalized by the CCITT in Geneva, November, 1980.)

## BRIEF HISTORY OF FACSIMILE

Hardly an article is written on facsimile without mentioning Alexander Bain's early work in 1842 using two swinging pendulums. However, the single most important contribution to this technology is not a technical breakthrough or years of research; it is standardization.

Facsimile is a communications device and, as such, is most useful when used as a graphic extension to voice communication. This requires standardization between equipments.

Through the impressive cooperative efforts of the PTT's (Postal Telephone and Telegraph Administration of each country) and the manufacturers working within the framework supplied by the CCITT (Consultative Committee International Telephone and Telegraph), a series of recommendations has been completed. These recommendations define the operation of the three groups of machines in use today.

- T.2—Defines Group 1 apparatus (6 minute FM)
- T.3—Defines Group 2 apparatus (3 minute AM)
- T.4—Defines Group 3 apparatus (approximately 1 minute V.27 ter)
- T.30—Defines the handshaking procedures for Groups 1, 2, and 3

The significance and applicability of T.4, T.30, and their interaction is the main topic of this paper. However, an understanding of the evolution and scope of these recommendations, particularly T.30, is essential to realizing their application.

## SCOPE OF T.30

The protocol and handshaking procedures recommended in T.30 for document facsimile transmission over the General Switched Telephone Network (GSTN) apply to facsimile service defined as Group 1, 2, and 3 type service. Machines providing these services should comply with CCITT Recommendations T.2, T.3, and T.4, respectively. These three groups of machines provide a wide range of service with a corresponding range of complexity and technology. Recommendation T.30 was designed to have the flexibility to meet the different needs of these machines.

In its simplest usage T.30 specifies a limited command/response repertoire to be implemented using a simple set of tones. When the service demands a more elaborate procedure, T.30 provides for a more extensive command/response repertoire implemented by modulating (at 300 bps) two of the tones previously specified (i.e., 1650 Hz and 1850 Hz). Thus the T.30 procedures allow any one or any combination of facsimile services to be incorporated in a particular facsimile unit.<sup>1</sup>

The CCITT also recognized that individual manufacturers will desire the design freedom to provide service in a manner unique to their market. Thus, Recommendation T.30 not only allows for a range of complexity but also allows a range of machine configuration (i.e., Simplex, Half Duplex, and Full Duplex). For those applications requiring even more design freedom, T.30 allows for special non-specified options.

Recommendation T.30 defines the line control procedures to provide three major functions. These functions are:

1. The establishment of a compatible operating mode between two facsimile units. This function is divided into two phases:
  - a) Phase A which defines the call establishment procedures for manual and automatic machines;
  - b) Phase B which utilizes a signaling scheme recognizably unique from the message signaling scheme to define the pre-message procedures which identifies and selects the proper operating mode.
2. The orderly means for transmitting and verifying a fac-

simile document(s). This function is divided into two phases:

- a) Phase C which utilizes the message signalling scheme for the actual transmission of the document (this phase is covered by the appropriate Recommendation for the equipment);
  - b) Phase D which defines the post-message procedures. Phase D specifies the end of Phase C, the status of the message received, and the next phase in the overall procedure.
3. The orderly means for terminating a facsimile call. This function is defined in phase E.

#### SCOPE OF T.4

Whereas T.30 defines the procedures necessary for document transmission, T.4 defines the specific apparatus characteristics relating to inter-operability of Group 3 apparatus. As is T.30, T.4 is concerned only with operation over the public switched telephone network.<sup>2</sup>

#### ESSENCE OF T.4

Since the CCITT and the EIA will both be publishing a detailed description of T.4, I shall endeavor to give the reader only the essential items contained in T.4 and perhaps some insight not available in the formal spec.

#### *Apparatus dimensions*

Scanning is accomplished left to right across a 215 mm line. Each line consists of 1728 elements. The vertical scanning density is 3.85 lpm with an optional 7.7 lpm higher resolution. COMMENTS: A4 paper is 210 mm wide. 8½" "U.S." paper is 216 mm wide.

#### *Minimum transmission time per scan line*

The standard minimum time is 20 msec with 40 msec, 10 msec, 5 msec, and 0 msec options. COMMENTS: This is basically a printer spec. It assures that data will not be sent faster than it can be printed. The 40 msec "option" was inserted to allow for a low-cost unit.

#### *One-dimensional coding*

One-dimensional run length data compression is accomplished by the popular modified Huffman scheme. In this scheme, black and white runs are replaced by a base 64 code representation. Compression is achieved since the code word lengths are inversely related to the probability of the occurrence of a particular run (reference Figure 1).

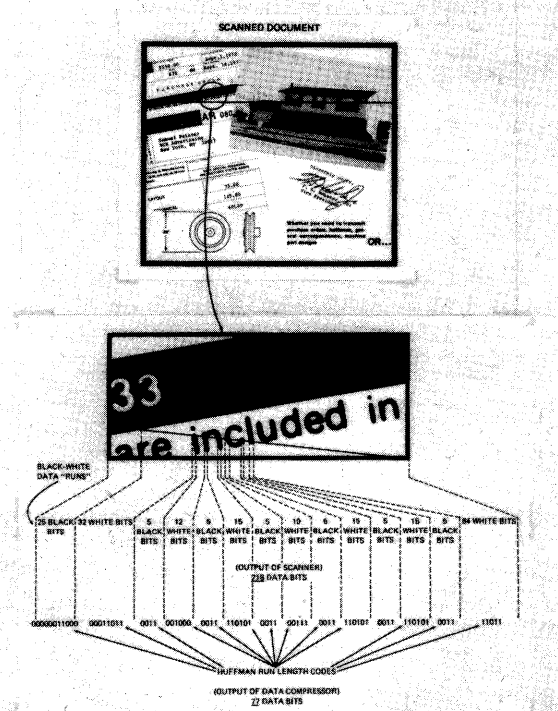


Figure 1—One-dimensional coding.

#### *Two-dimensional scheme*

The two-dimensional coding scheme agreed upon by the CCITT in November, 1979 is labeled as the Modified READ Code. It is an algorithm based on the original Japanese READ (Relative Element Address Designate) Code but modified considering U.S.A. concerns on the algorithm's performance and ease of implementation. The following principles were incorporated into the modified READ code.

#### **One-dimensional EOL code plus a tag bit**

This code offers increased immunity to noise by improving error detection and recovery, minimizes additional cost as the hardware already exists to detect this code for the one-dimensional standard, simplifies the injection of the FILL code (as it is a natural extension of EOL), and eliminates the need to provide zero insertion (bit stuffing) into the data stream.

#### **A maximum standardized K value with optional programmability of smaller values**

The ability to provide for programmability of the parameter K improves compression efficiency by choosing either the one-dimensional or two-dimensional code on a scan-line by scan-line basis. Additionally, where the document content is such that a large amount of FILL is present in the

data stream, the one-dimensional code can be chosen to improve error performance. The maximum value of  $K$  shall be set as follows: normal resolution (i.e., 3.85 lpm), with  $K$  less than/equal to 2; high resolution (i.e., 7.7 lpm), with  $K$  less than/equal to 4.

#### **An option for an uncompressed mode within a scan-line of data**

The two-dimensional code was chosen to have the ability to be extended to an uncompressed mode of operation. This avoids expansion of the compressed data for certain document conditions. NOTE: Although the eight test documents chosen for evaluating these coding schemes do not exhibit this effect, certain documents will.

#### **Truncate examination of the history line in the vertical mode of less than/equal to $\pm 3$ pels**

Examination of the data set forth to date indicates that examination of the history line beyond  $\pm 3$  pels offers little or no improvement in compression efficiency and complicates the implementation of the coding scheme. COMMENTS: The resultant code bears a strong resemblance to the U.K. R2 code which was itself a modified READ code but more in line with the above-mentioned principles. Extensive testing on the U.K. code showed it to be slightly better than the previously unmodified READ code.

Two-dimensional coding is most efficient at high resolution and only marginally efficient at the standard resolution.

#### *Modulation and demodulation*

There are at present four data rates defined for Group 3 apparatus: standard data rates of 4800 bps and 2400 bps in accordance with CCITT Recommendation V.27 ter and optional data rates of 9600 bps and 7200 bps in accordance with V.29. COMMENTS: It was generally felt that 4800 and 2400 bps would apply to the Public Switched Telephone Network (PSTN) and that 9600 and 7200 bps would apply to private networks. However, it is clear that the Recommendation allows for any speed operation over either type of circuit provided the performance of the service is not degraded.

#### **COMMENTS ON T.4**

The standard mode of operation is one-dimensional modified Huffman coding at normal resolution, 20 msec per scan line, transmitted at 4800 bps, using the V.27 ter modulation process.

Selection of the various other modes of operation can be done during the initial handshaking or between documents. The details of this procedure are contained in Recommendation T.30.

Although there is no standard algorithm for selecting optional capability, it is generally understood that: the transmission rate is selected as high as possible while still maintaining acceptable copy quality; one-dimensional coding is best for normal resolution; two-dimensional coding is best at high resolution; and the capabilities of the printer set the minimum scan line time.

#### **ESSENCE OF T.30**

As mentioned previously, T.30 is concerned with the *procedures* which are necessary for document transmission between two facsimile stations in the general switched telephone network. These procedures essentially comprise the following:<sup>3</sup>

1. call establishment and call release;
2. compatibility checking, capability status, and selection command;
3. checking and supervision of phone line conditions;
4. control functions and facsimile operator recall;
5. both recognized optional functions as well as other (non-standard) options.

The above-mentioned features can be accomplished, at best in a simplified manner, by the use of tones. And indeed, the tonal procedures are very popular for Group 2 equipment. For more sophisticated automatic equipment and for all Group 3 equipment, the binary coded procedures (300 bps) are used. The emphasis of the binary coded procedures is in the compatibility/capabilities checking/selecting feature. This feature is implemented in a straightforward manner. One unit (the identifying unit) lists all of its capabilities, both standard and optional, and the other unit (the commanding unit) selects from that list the most appropriate mode of operation. Understanding this simple feature is the key to understanding T.30.

The first phase in T.30 (Phase A) is to establish a telephone list, then decide which unit will be the identifying unit and which will be the commanding unit. This rule is followed: all manual receivers and all auto answer units are identifying stations; all manual transmitters and all auto dialing units are commanding stations. NOTE: Calling an auto answer transmitter is a polling operation.

The next phase (Phase B) contains the important pre-message procedures. It encompasses both tonal and digital operation. In the procedure, an attempt is made by the identifying unit to initiate either the digital procedures (by transmitting the Digital Identification Signal, DIS) or the tonal procedures (by transmitting the appropriate Group Identification, GI, tone) (reference Figure 2).

The commanding station analyzes these signals, then initiates either the digital procedures (by sending the Digital Command Signal, DCS, followed by modem training) or the tonal procedures (by sending the appropriate Group Command, GC, tone followed by Phasing). The identifying sta-



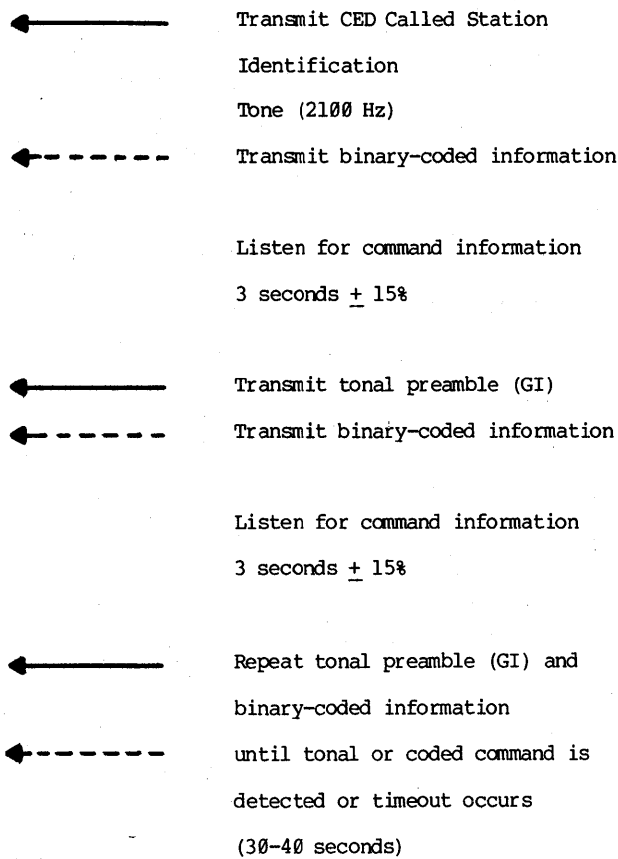


Figure 2—Initial identification attempting both tonal and digital handshaking.

tion assumes the selected mode, processes the training (or phasing) and, if all is OK, responds with a Confirm to Receive, CFR, reply. Phase C message procedure is now ready to begin.

Phase C is referred to only briefly in T.30 since this phase is covered in detail in the appropriate T.2, T.3, or T.4 Recommendation.

Phase D, the end-of-message phase, serves to verify the successful (or unsuccessful) reception of the document and to direct the procedure to the next appropriate phase (usually either Phase C or Phase E).

Phase E is simply Call Termination, where the control is returned to the phone handset (this usually implies disconnection if the handset is on hook). Figure 3 shows a signal sequence diagram of the five phases.

To handle line errors, T.30 incorporates a method, similar to HDLC, wherein commands are repeated (three times) if not responded to with valid responses. Accordingly, commands which are received in error are disregarded.

**OPTIONS WITHIN T.30**

Recommendation T.30 is rich in options giving the latitude and flexibility to form a family of products while still maintaining overall compatibility. Whereas these options pertain

to all three groups of apparatus, they are perhaps most appropriate and best understood in reference to Group 3 equipments.

Options fall into two categories: recognized options, where the operation is internationally defined; and non-standard options which are known only to a particular machine model or communication service.

The following are recognized options which are defined within the Digital Identification Signal, DIS. Their operation is either self-explanatory or defined in Recommendation T.2, T.3, or T.4. These options relate to apparatus parameters.

1. Group 1, 4-minute operation with an Index of Cooperation of 176.
2. Group 1, 2, and 3 capability within one physical unit (reference T.2, T.3, T.4).
3. 9600 and 7200 bps operation in accordance with V.29 (reference T.4).
4. Higher resolution (7.7 lines/mm in the vertical).
5. Two-dimensional coding (reference T.4).
6. Wider paper widths (256 mm, B4, and 297 mm, A4).
7. Longer paper lengths (364 mm, B4, and unlimited, roll feed).
8. Additional scanning times (faster: 10, 5, and 0 sec; or slower: 40 msec).

T.30 also includes recognized options relating to procedure. These are:

*Station identification*

Three optional commands allow for identification of the transmitter and/or the receiver using a 20-digit field. This field contains the international telephone number.

*Voice request*

Three optional commands allow an operator at either end to interrupt facsimile procedure and request voice contact.

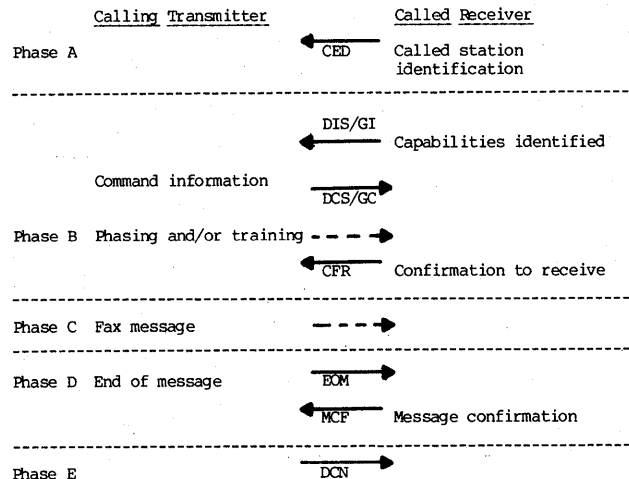


Figure 3—Calling station transmitting.

### Improved error detection

An optional response can be the response to a corrupted command which shortens the 3-sec delay before the command is repeated.

### 2400 bps operation of T.30

The 300 bps FM (1650 Hz/1850 Hz) modulation of T.30 was deliberately chosen as a unique modulation process identifiably different from the modulation used for the facsimile message. This uniqueness avoids confusion when shifting from one message data rate to another. However, for low-cost machines or for applications where only one fixed speed is desirable, T.30 can operate at the message rate of 2400 bps.

In addition to these recognized options, manufacturers or

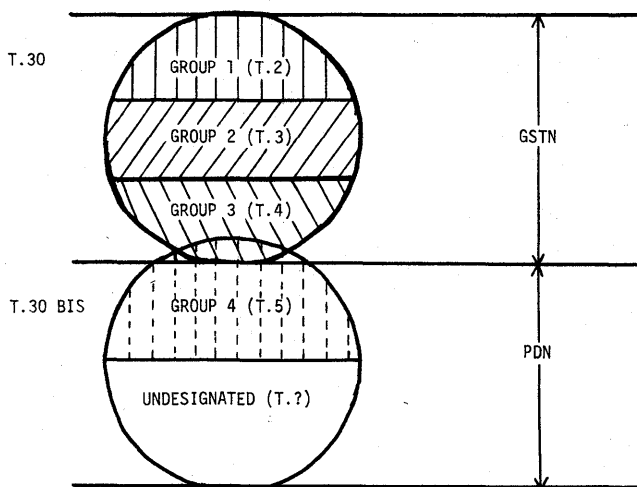


Figure 4—Relationship between facsimile service, facsimile standards, and public communication facilities.

service groups can offer unique features by utilizing the non-standard facilities commands/responses. These signals contain a unique, 16 bit, registered, manufacturer's code identifying that signal as pertinent only to one manufacturer. An example of a non-standard option would be scrambling, or an alternative data compression scheme.

### FUTURE WORK

Recommendation T.30 was developed to meet the present demand for Group 1, 2, and 3 facsimile service over the GSTN and in this context it is an optimal procedure. A new group(s) of service is now being discussed which could take advantage of the newly developing Public Data Networks (PDN) and which could offer some very sophisticated features to the customer. This "Group 4" service has only been briefly discussed in CCITT meetings, but it is obvious that it will require a new machine recommendation (e.g., T.5) and a new procedure definition (e.g., T.30 bis).

An overview of these services and the related specifications is graphically depicted in Figure 4.

The details of the interplay between Group 3 service and Group 4 service will be an item for close study. And it should be noted that whereas Recommendation T.30 bis will have a broader scope of application, it is not true that Recommendation T.30 will be outdated by T.30 bis or in any way conflict with T.30 bis. Rather, the two recommendations will complement each other providing a total range of cost effective service.

### REFERENCES

1. CCITT COM XIV—No. 47-E. Source: United States of America. Title: Treatise on Recommendation T.30 (Revised).
2. CCITT COM XIV—D21. Source: United States of America. Title: Proposed U.S.A. Standardization of "Group 3 Facsimile Apparatus for Document Transmission in the General Switched Telephone Network".
3. CCITT COM XIV—D20. Source: United States of America. Title: Proposed U.S.A. Standardization of "Procedures for Document Facsimile Transmission in the General Switched Telephone Network".



# The application of optical character recognition techniques to bandwidth compression of facsimile data

by PATRICE J. CAPITANT and ROBERT H. WALLIS

*Compression Labs*  
Cupertino, California

## INTRODUCTION

The goal of facsimile bandwidth compression is the efficient transmission of documents achieved by the removal of redundancy in the encoding technique. For the case of printed or typewritten documents, the most powerful encoding technique is the Combined Symbol Matching (CSM) algorithm, which is based on the detection of recurrent patterns (such as alphanumeric characters) in the document being encoded (4,5,6,10). As the transmitter scans the document, it locates and extracts isolated patterns, transmits them to the receiver, and stores them in a library. Using the received patterns, the receiver also accumulates an exact copy of the transmitter's library. As each new pattern is isolated, it is compared with the library patterns which have been previously encountered. If the pattern is unfamiliar, it is added to the library. However if a "match" is detected, this indicates a recurrence of a pattern, and there is no need to retransmit it, since it is available in the receiver's copy of the library. Therefore, the library entry number (library ID) is transmitted instead, enabling the receiver to reconstruct the pattern from the "prototype" in its library. Since the library ID can be transmitted with far fewer bits than the binary pattern that it points to, a significant bandwidth compression may be attained. For printed documents, the CSM algorithm is typically twice as efficient as the best run-length coding algorithms.

In order to operate efficiently, the transmitter must not allow very many recurrences to escape detection, since this leads to a loss of compression. Conversely, it must also avoid declaring dissimilar characters to be a match, since this leads to a substitution error in the reconstructed document. This paper deals with the way in which the CSM algorithm determines whether two patterns match or not, and how the basic algorithm may be modified to perform optical character recognition.

## COMBINED SYMBOL MATCHING SYSTEM

The Combined Symbol Matching (CSM) system is a dual mode encoding system that possesses the advantages of extended run-length encoding and symbol recognition systems.

Figure 1 illustrates the block diagram of the encoding system. In operation, a number of scan lines (equal to about two to six times the average character height) of binary image data are stored in a scrolled buffer. This data is then examined line-by-line to determine if a black pixel exists. If the entire line contains no black pixel, the information is encoded by an end of line code. On the other hand, if a black pixel exists, a blocking process is conducted to block the symbol. For those blocked symbols, further processing is required to determine if a replica of the symbol in question already exists in the library. This process involves the extraction of a set of features, a screening process to reject unpromising candidates, and finally a series of template matches. The first blocked symbol and its feature vector are always put into the prototype library, and as each new blocked symbol is encountered, it is compared with each entry of the library that passes the screening test. If the comparison is successful, the library identification (ID) code along with the location coordinates are transmitted. If the comparison is unsuccessful, the new symbol is both transmitted and placed in the library. Those areas in which the blocker cannot properly block the symbol are assigned to a residue, and a two dimensional run-length coding technique is used to code the residue data.

The following sections summarize the compression and expansion algorithms.

### *Compression*

The compression technique is based on the following sequence of operations.

1. The raster image is scanned, one line at a time, until a black pixel is found. This is called a "key pixel."
2. The local area around the key pixel, called a "trial block," is examined to isolate a small contiguous group of black pixels roughly the size and shape of an alphanumeric character.
3. If no symbol is found within the trial block, the trial block is left as a residue. The blocker is designed to avoid the residue in subsequent search for key pixels.
4. Residues are encoded using a two dimensional run-length code (8,9).

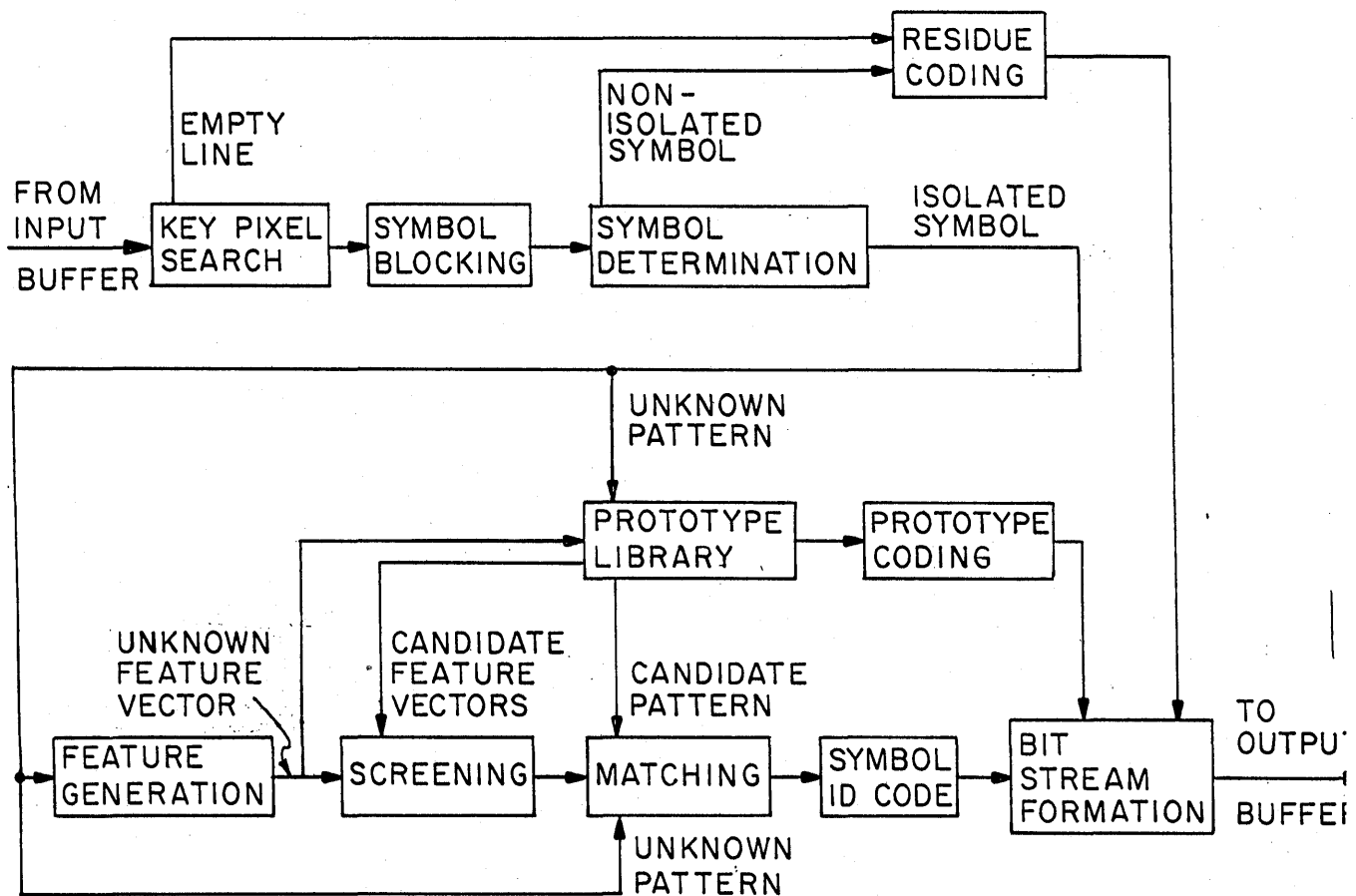


Figure 1—Combined symbol matching facsimile coder.

5. Symbol blocks are isolated, labeled, and removed from further key pixel consideration. The blocks are used to build a library against which other symbol blocks may be matched. The library is built from scratch with each new page, and updated as new symbols are found.
6. A set of features is measured for each symbol block. These are also stored in the library.
7. As each symbol is found, its features are compared to the features of symbols in the library. Then library entries with features most like the new blocks are treated as candidates for the matching process.
8. Library candidates are aligned with the trial symbol block, and a pixel-by-pixel comparison is made.
9. If the matching error is less than a threshold, the matching process is stopped, and the identification of the matching library member is stored for later coding.
10. If no match is found, the block is stored in its entirety as a "prototype" block. It is also entered into the library.
11. When a line has been completely processed, i.e., all pixel patterns in the line have been labeled as either residue or symbols, the code bits for the residue are concatenated with the code bits for the symbol blocks,

a linesync code word is added every  $K$  lines, and the resulting sequence is transmitted as the data stream.

#### Expansion

The expansion technique is based on the following sequence of operations.

1. Portions of the input data stream carrying the code for one line of the image are isolated. The lines are separated by special "end of line" code words.
2. The code format is used to separate residue from the symbol code. These codes are processed separately.
4. The symbol code is used to generate a library of symbol blocks, and the ID code is used to select library entries.
5. The residue and symbol blocks are combined.

#### System elements

The following sections describe the key elements of the CSM system.

### Input Buffer

The facsimile data to be processed is stored in a scrolled buffer that "scrolls" through the input document. This is accomplished by rotating the addressing of the input memory such that the newest line that enters the buffer is written over the oldest line. The buffer contains 128 lines, which is about four times the height of the largest character that can be matched.

### Symbol blocking

The function of the key pixel scanner Symbol Blocker is to examine the input buffer in a systematic fashion, and to locate the position and size of any isolated symbols. A pixel in the buffer, denoted here by the character '@', is considered to be a *key pixel* whenever it is black and the four neighbors located above it and to its left are white, as shown.

...  
.@

Whenever a key pixel is encountered, the blocker is initiated. It will try to extract the symbol connected to the key pixel at its top and is delimited by a border of white pixels. If such a symbol can be found, it has to fit with its border into a  $32 \times 32$  array. If no connection is made, go to the next key pixel. If a connection is made, the symbol is stored with its borders in a  $32 \times 32$  RAM and erased from the input buffer.

The blocker algorithm separates the symbol from its surroundings by determining its boundary (perimeter). It does this by starting on a known exterior point (i.e. the key pixel) and following the exterior of the symbol in (say) the clockwise sense until the entire perimeter has been traced. Consider the eight nearest neighbors of a central pixel to be indexed as follows:

5	6	7
4	X	0
3	2	1

A clockwise rotation from a perimeter point ( $i$ ) is given by

$$j = (i + n) \pmod{8}$$

where  $n$  is the number of clockwise 45 degree increments in the rotation. The perimeter following algorithm consists of searching through the boundary points in a clockwise sense until a black pixel is encountered. The search is initiated at the neighbor that corresponds to the previous boundary point.

Assuming "X" in the above diagram is the key pixel of a possible symbol in a trial block, the algorithm first rotates counter-clockwise from index four until a black pixel is encountered. This vector is stored, and utilized as the stopping criterion, since the last vector in the clockwise trace is exactly 180 degrees out of phase with the first vector in a counter clockwise trace. In terms of the above notation

$$\text{LAST(CLOCKWISE)} = \text{FIRST(COUNTER-CLOCKWISE)} - 4 \pmod{8}$$

Once the perimeter has been traced, the character must be extracted from the page for processing. This is accomplished by generating a "mask" which contains all the interior points of the perimeter, and performing a Boolean "and" between the mask and the trial block. The mask may also be used to erase the processed character from the document so that the residue contains only unblocked patterns.

The generation of the mask is based on the relation between a closed curve and its internal area. Specifically, the line integral gives:

$$A = \oint y dx$$

where  $A$  is the internal area.

A discrete counterpart of the continuous line integral expression has been developed which is amenable to digital mechanization. Each link of the perimeter is specified by a chain code in the range (0,7), and thus representable as a 3 bit code. The following algorithm, which starts with a *blank* buffer generates a mask of all the interior points of the perimeter:

- (1) Complement all locations to the left of the key pixel and use the vector whose destination is the key pixel as initial source vector.
- (2) All locations are then determined by a source vector and a destination vector. If  $N(s) = A \times 4 + B \times 2 + C$  is the direction of the source vector (bits  $A, B, C$  specify the chain code) and  $N(d)$  is the direction of the destination vector, let  $N(d) - 1 \pmod{8} = D \times 4 + E \times 2 + F$ . Let

$$\begin{aligned} R &= A \text{ .nor. } D \\ L &= A \text{ .and. } D \\ S &= (.not. E \text{ .or. } B) \text{ .and. } (A \text{ .xor. } D) \end{aligned}$$

Apply the following rules:

$$\begin{aligned} R = \text{.true.} &\Rightarrow \text{Complement location and locations to the left} \\ L = \text{.true.} &\Rightarrow \text{Complement locations to the left} \\ S = \text{.true.} &\Rightarrow \text{Complement location} \end{aligned}$$

After the entire perimeter has been followed the mask buffer is complete.

In addition, the area enclosed by the boundary may be easily calculated as the perimeter is being traced. It is given by

$$\text{AREA} = \sum_{J=0}^{N\text{LINKS}-1} \text{COL}(J) [\text{ROW}(J+1) - \text{ROW}(J)]$$

where  $N\text{LINKS}$  equals the number of lines in the chain of coordinate pairs representing the perimeter, and all indices are taken modulo  $N\text{LINKS}$ .

The area enclosed by the perimeter has proved to be a useful feature for symbol recognition. Specifically, the ratio of the perimeter squared to the area is invariant to magnification, rotation, and translation.



sity" of every black pixel is merely the sum of all the pixels in its  $3 \times 3$  neighborhood if the pixel is 1, and 0 if the pixel is 0. The patterns above labelled "Weighted XOR Error" were calculated in this manner. Note that by this criterion, the associated counts indicate that the pair "c" and "o" are more separated (Count=131) than the pair of "e's" (Count=73).

### Optical Character Recognition

The CSM system can be modified to perform optical character recognition and various hybrid CSM/OCR tasks.

If a fixed set of symbols is to be expected, the library can be preconstructed. Unrecognized symbols would then be transferred in the residue.

A further modification would remove the residue coding subsystem and only transmit recognized symbols and their position on the page.

Finally the OCR mode would not transmit positions but only library codes, in the sequence which they would be read. In this mode, however, two extra subsystems need to be implemented to allow a meaningful reproduction.

### Line Tracking

In the Western world, printed matter is "read" from left to right, and from top to bottom. Therefore, a symbol blocking system that transmits its output to a serial ASCII terminal must do the same. However, the CSM algorithm extracts

characters from the document being scanned in a totally different fashion. As the line buffer scrolls through the page from top to bottom, the tallest of first encountered characters are removed from the document and processed through the recognition algorithm. Thus, characters emerge from the CSM process in a sequence which would be totally incomprehensible if viewed in chronological sequence. In the conventional CSM facsimile transmission mode, this is of no consequence, since characters are placed in their appropriate address locations regardless of their order of occurrence. In the serial symbol recognition mode, the transmitter will assign each character an ASCII code, assemble the codes into lines, inserting blanks, line-feeds, carriage returns, etc., and transmit the lines serially to the receiver. For single spaced or rotated documents, this "line-tracking" is more difficult than one would imagine. The problem is basically that of grouping the characters into lines. Determining the sequence in which they should be transmitted is relatively easy since the characters may be sorted by their column addresses. A significant benefit of this serial ASCII mode is that no information on character location need be transmitted since the correct sequence is all that is required in order to properly reconstruct the received document.

The line tracking algorithm is based on a straight line fit of the key pixel coordinates of characters on a text line, as illustrated in Figure 2. The straight line is defined parametrically as

$$R = S \times C + O$$

where  $R$  represents the row index,  $C$  is the column index,  $S$  denotes the text line slope, and  $O$  is its offset. As characters are encountered, they are assigned to the nearest straight line representing a text line.

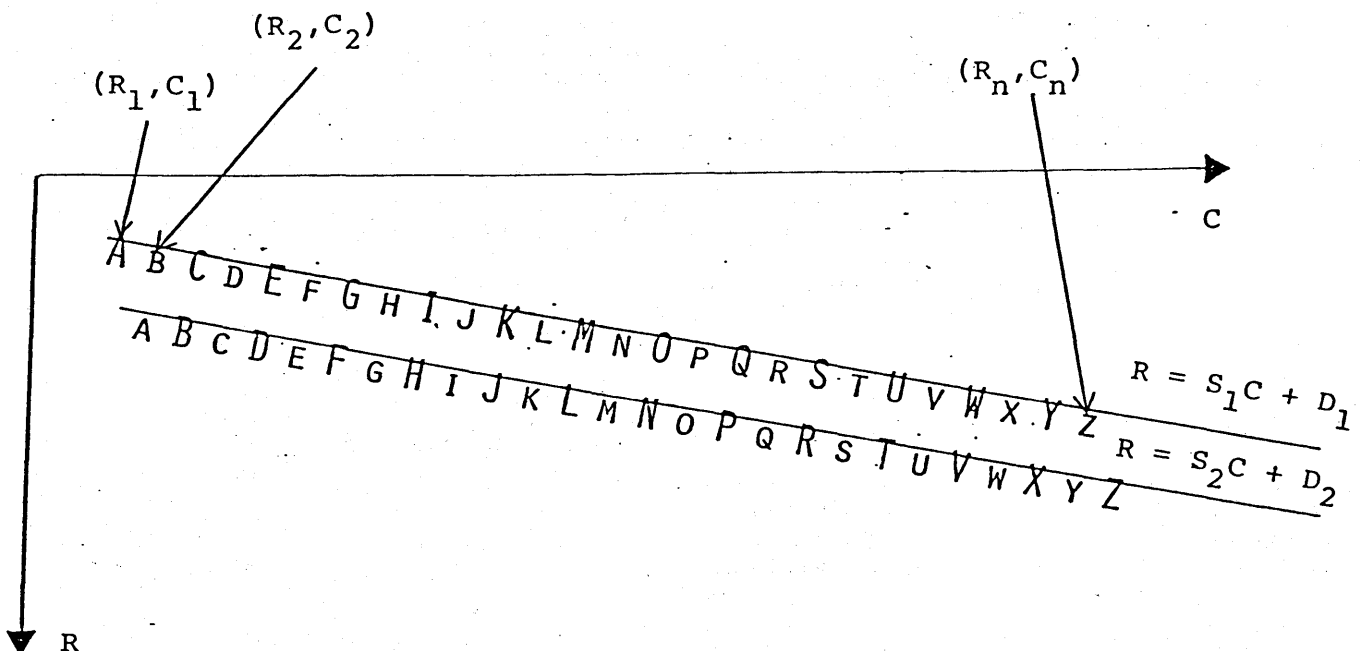


Figure 2—Line tracking.





COMPRESSION LABS, INC.

August 15, 1978

Telecommunications Manager  
International Company  
1111 Broadway  
New York, N.Y. 10022

Dear Mr. Manager:

This letter will act as the standard for determination of the minimum compression ratios acceptable for the FAX-COMP, facsimile data compressor. The floppy disk of the FAX-COMP will be able to store at least nine copies of this page prior to overflowing which will guarantee a transmission time of less than 25 seconds for the page. This transmission time will be achievable using a 2400 baud digital modem for line connection.

Compression ratios of from 5:1 up to 25:1 can be expected from other pages of information, depending upon the actual content of the pages. These compression ratios are defined when using the 96 line per inch scanning resolution only.

Very truly yours,

CLOYD E. MARVIN

Vice President

Marketing

CH:vg

August 15, 1978

Telecommunications Manager  
International Company  
1111 Broadway  
New York, N.Y. 10022

Dear Mr. Manager:

This letter will act as the standard for determination of the minimum compression ratios acceptable for the FAX-COMP, facsimile data compressor. The floppy disk of the FAX-COMP will be able to store at least nine copies of this page prior to overflowing which will guarantee a transmission time of less than 25 seconds for the page. This transmission time will be achievable using a 2400 baud digital modem for line connection.

Compression ratios of from 5:1 up to 25:1 can be expected from other pages of information, depending upon the actual content of the pages. These compression ratios are defined when using the 96 line per inch scanning resolution only.

Very truly yours,

?

CLOYD E. MARVIN

Vice President

Marketing

CM:vg

Figure 3—OCR results.

### Handling of Special Characters

A number of characters which consist of two "sub-characters" must be treated as special cases in the symbol-matching mode. This is because the blocker/matcher would otherwise fragment them into their constituent parts and give misleading results. These characters are: (i), (j), (!), (:), (;), (=), and ("). After recognition of the two parts of the character, the system will check if two compatible symbols are on top or almost on top of each other. If so, the two symbols are merged into one. For example, two (.)'s on top of each other will be merged into a (:).

### Performance

The CSM symbol recognition system has been extensively evaluated by computer simulation to optimize its performance and to determine its compression ratio with respect to other coding methods. The symbol recognition mode system has been tested with 86 sets of data, each containing 1,000 samples of one of the 86 symbols of the Courier 10 font. In these tests, no mismatches occurred, and only very badly damaged characters were rejected.

Figure 3 contains an example of a business letter and its reconstruction in the symbol recognition mode of operation. It should be noted that the reconstructed letter has been printed with a different font than the original; however, the format and spacing of the two letters are in basic agreement. The compression factor obtained for this document for operation of the CSM system in the symbol matching mode is about 257:1 and for operation in the facsimile mode is about 49:1.

### REFERENCES

1. Arps, R. B., "Binary Image Compression," in *Image Transmission Techniques*, W. K. Pratt, Ed., Academic Press, New York, 1979.
2. Musmann, H. G. and Preuss, D., "Comparison of Redundancy Reducing Codes for Facsimile Transmission of Documents," *IEEE Transactions on Communications*, Vol. COM-25, No. 11, Nov. 1977, pp. 1425-1433.
3. Ascher, R. N. and Nagy, G., "A Means for Achieving a High Degree of Compaction on Scan-Digitized Text," *IEEE Transactions on Computers*, Vol. C-23, No. 11, November 1974, pp. 1174-1179.
4. Pratt, W. K., Chen, W., and Reader, C., "Block Character Coding," *Proceedings SPIE*, August 1976, pp. 222-228.
5. Chen, W., Douglas J. L., and Widergren, R. D., "Combined Symbol Matching—A New Approach to Facsimile Data Compression," *Proceedings SPIE*, August 1978, pp. 2-9.

- 
6. Chen, W., Douglas, J. L., Pratt, W. K., and Wallis, R. H., "A Dual Mode Hybrid Compressor for Facsimile Images," *Proceedings SPIE*, August 1979.
  7. White, H. E., Lippman M. D., and Powers, K. H. "Dictionary Look-Up Encoding of Graphics Data," in *Picture Bandwidth Compression*, T. S. Huang and O. J. Tretiak, Eds., Gordon and Breach, New York, 1972, pp. 265-281.
  8. Mitchell, J. L. and Goertzel, G., "Two-Dimensional Facsimile Coding Scheme," *Proceedings International Communications Conference*, 1979, pp. 8.7.1-8.7.5.
  9. "Proposal for Draft Recommendation of Two-Dimensional Coding Scheme," *Report of Study Group XIV*, Contribution No. 42.
  10. Pratt, W., Capitant, P., Chen, W., Hamilton, E., and Wallis, R., "Combined Symbol Matching Facsimile Data Compression System," *Proceedings IEEE*, May 1980.



# Facsimile image coding

by JOAN L. MITCHELL

IBM Thomas J. Watson Research Center  
Yorktown Heights, New York

## INTRODUCTION

Facsimile image coding has recently received a lot of attention because of the standardization work being done in this area by the International Telegraph and Telephone Consultative Committee (CCITT). In November, 1977, CCITT Study Group XIV standardized a one-dimensional data compression scheme for facsimile images.<sup>1</sup> Two years later this standard was incorporated in a recommended two-dimensional coding scheme.<sup>2</sup>

Data compression (coding) of images is essential if high quality digital facsimile is to be practical.<sup>3,4</sup> A standard facsimile document has 3.85 scan lines/mm vertically. However, an optional vertical resolution of 7.7 lines/mm can be used for images with fine details. The standard horizontal resolution is fixed at 8.04 picture elements/mm (1728 picture elements in 215 mm) to avoid excessive "stair stepping" on curved or diagonal edges of the reproduced images.

Almost a quarter million bytes are required to store the raw digitized data of a 215 mm × 280 mm page at the normal resolution. At the higher resolution, close to a half million bytes are needed. Run length coding schemes, such as the CCITT one-dimensional standard, reduce the storage requirement by factors of 5 to 15 for typical images.

Two-dimensional schemes take advantage of correlation between successive scan lines to obtain better compression.

This paper considers only black/white reversible facsimile coding. The original black/white images are reconstructed exactly (assuming no errors during transmission or storage) after each encoding/decoding cycle.

## SOURCE MODEL

The facsimile scanned image can be described mathematically as a set of symbols representing various patterns of picture elements (pels). The set of symbols, the "source model," must be sufficiently general that all possible configurations of pels can be described. Each possible set of symbols has particular statistical properties; once these statistical properties are understood and the coding rules defined, the generation of variable length code words can be a straightforward application of Huffman<sup>5</sup> coding. Normally, the tables are selected for a large set of documents and therefore may not be optimum for individual documents. Shan-

non's entropy,<sup>6</sup> the minimum number of bits per symbol, together with the number of coded symbols can be used to calculate the lower bound on the number of bits needed to code each document for a given model. The entropy is a measure of the "randomness or surprise" found in an image for a particular model. Thus, the entropy of a document can change dramatically when a new source model is used.

Fixed codes assume that average statistics apply to all documents. However, some documents do not fit these statistics. When the assumptions are completely invalid, regions of the "compressed" image may require many more bits than the original uncompressed image. The uncompressed mode option in the two dimensional CCITT recommendation<sup>2</sup> can be used to minimize the worst case local expansion.

## ONE-DIMENSIONAL CODING

The black and white picture elements in typical images are not randomly distributed but come with a high degree of correlation or redundancy. In each horizontal scan line the pels of a given color (black or white) tend to come in groups or runs. Run length coding takes advantage of this horizontal correlation in a practical way.<sup>7,8</sup> The number of pels between color changes is coded instead of each pel. The runs alternate in color so that as long as the first run in a line is guaranteed to be a white run no extra bits are needed to specify the color. A length of zero can be used if the line actually starts black. Figure 1 shows some examples of run lengths.

An average entropy per pel for run length coding was obtained by taking eight CCITT documents scanned at 1728 pel/line for 2376 lines (nominally 8 × 8 pel/mm) and collecting the statistics of the entire set together. To simulate the standard resolution the odd numbered lines were used (nominally 8 × 4 pel/mm). From the run length probability distribution a lower bound of 0.113 bits/pel was obtained for both resolutions (HIGH and STD). This entropy per pel for mixed black and white run lengths is used as a reference point and therefore is entered in Table I as 100 percent.

The model which considers the black and white run lengths as the same statistically is not optimum. It is well-known that a strong peak in the black run distribution<sup>9</sup> usually occurs for runs of two or three pels for typewritten documents scanned at this resolution. The white runs tend to have a

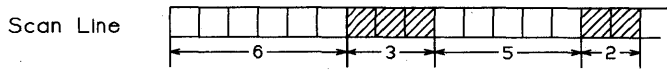


Figure 1—Run length examples.

less sharp peak spread out over the 4 to 7 pel runs. Optimizing the black and white runs separately (B & W Sep.) decreases the entropy per pel by a few percent.

*CCITT Modified Huffman code*

The CCITT one-dimensional standard Modified Huffman Code<sup>1</sup> is a run length coding scheme in which the black and white runs have separate tables. Runs longer than 63 are coded in two pieces in order to decrease the size of the code tables by an order of magnitude. Multiples of 64 are coded first as a Makeup Code and then the remainder follows as a Terminating Code. Short runs (0-63) only need the Terminating Code.

Since the CCITT standard is for telephone transmission, the standard also includes a unique end of line (EOL) code for resynchronization after transmission errors. It is a unique pattern of eleven zeros followed by a one. Extra zeros can precede the EOL as fill bits to maintain a minimum transmission time per line.

VERTICAL REFERENCE CODING

Run length coding can only take advantage of the horizontal redundancy. There is also strong vertical correlation in most images which comes from the vertical continuity of objects, strokes, or lines. Vertical reference coding<sup>3,4,10-13</sup> codes a run as the difference between the run length and the distance to the same color change on the history line (preceding scan line). Figure 2 shows the black run ending one pel to the right of the first black-to-white change on the his-

TABLE I.—Entropy Per Pel as a Percent of One-Dimensional Mixed Run Length Entropy Per Pel (0.113 bits/pel)

RES.	MIXED		B & W Sep.		Pass, B&W Sep.	
	HIGH	STD	HIGH	STD	HIGH	STD
1-D	100.0	100.0	96.3	96.3	96.3	96.3
N=0	72.2	81.2	70.5	75.6	70.2	74.6
N=1	57.9	72.2	57.1	70.8	55.5	68.4
N=2	57.0	71.1	56.4	70.0	54.3	66.9
N=3	57.1	71.1	56.6	70.2	54.2	66.8
N=4	57.3	71.2	56.8	70.6	54.4	67.0
N=5	57.6	72.0	57.1	71.1	54.6	67.3
N=1 +	55.8	69.7				
N=3 +	54.8	68.9				

+ Conditional entropy

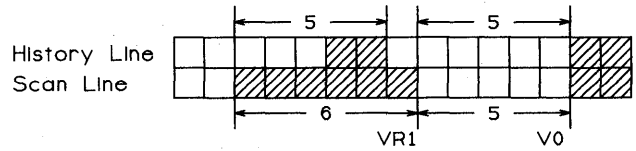


Figure 2—Vertical reference coding.

tory line (VR1). The white run has a vertical difference of zero (V0).

The entropy per pel as a percent of the one-dimensional entropy per pel is listed in Table I for various maximum allowed differences, *N*. Separating the black and white run statistics decreases the entropy per pel by a few percent relative to the combined statistics.

Generally about half of all the runs vertically align exactly and another 25 percent are within one pel of the appropriate transition. Larger vertical differences are much less frequent. Table II gives the distribution of runs in percent for *N*=3. Runs which stop to the left of the transition on the preceding line are given as VL*x* and those to the right are shown as VR*x*. For differences greater than *N* a run length (RL) code is used. The white and black runs have a similar distribution.

So far only independent statistics have been considered. Conditional probabilities can also be used. Table II illustrates how the distribution changes from almost 50 percent

TABLE II.—Distribution of Runs in Percent

RES.	VL3	VL2	VL1	V0	VR1	VR2	VR3	RL	PM
WHITE RUNS									
HIGH	0.8	2.9	16.0	46.9	15.0	2.4	0.7	15.4	
STD	1.9	4.9	11.4	40.1	10.4	4.0	1.5	25.7	
BLACK RUNS									
HIGH	0.8	2.8	16.3	43.6	15.7	2.8	0.9	17.2	
STD	1.8	4.0	10.4	37.3	10.7	4.1	2.1	29.7	
MIXED RUNS									
HIGH	0.8	2.9	16.2	45.2	15.3	2.6	0.8	16.3	
STD	1.8	4.5	10.9	38.7	10.6	4.1	1.8	27.7	
RUNS FOLLOWING V									
HIGH	0.7	2.8	17.3	49.3	16.6	2.8	0.8	9.7	
STD	1.7	4.7	12.4	45.9	11.6	4.5	1.9	17.3	
RUNS FOLLOWING RL									
HIGH	1.3	3.4	10.4	24.3	8.7	1.4	0.5	50.1	
STD	2.1	3.9	7.0	20.0	7.8	3.0	1.3	54.9	
WITH PASS MODE									
HIGH	0.9	2.9	16.8	46.9	16.1	2.7	0.7	12.9	5.6
STD	2.0	4.9	11.9	40.8	11.3	4.4	1.7	23.0	9.2

V0 to more than 50 percent RL depending upon whether the last code was a vertical (V) code or a run length (RL) code. The entropies calculated from the conditional probabilities for  $N=1$  and  $N=3$  are also given in Table I.

*IBM coding scheme*

IBM proposed to the CCITT a coding scheme<sup>14,15</sup> that only considered vertical transitions within one pel. (Most of the compression improvement attributable to vertical reference coding is obtained with  $N=1$ .) Figure 3 shows coding examples using vertical reference codes within one pel of history transitions.

Compression performance improvements are obtained by switching the code tables so that they depend upon whether the last code was a run length or a vertical reference code to take advantage of the differences in the conditional probabilities. The RL-prefix is followed by the run length count coded with the Modified Huffman codes. The code table is given below.

Code Table		
	Following RL	Following V
RL-prefix	1	01
V0	01	1
VL1	001	001
VR1	000	000 0001*

\* Even numbered VR1's in a series have an extra 1 to break up long strings of zeros.

Additional compression is obtained by removing redundancy from the code table. Pels at distances which would be coded with vertical reference coding if they were the final pel are not included in the run length count. A few percent can be saved by sometimes being able to represent a run by a count shorter than its true length.

*CCITT Modified READ code*

The recommended CCITT two-dimensional Modified Relative Element Address Designate (Modified READ) coding scheme<sup>2</sup> is a simplification of the original READ coding scheme<sup>16</sup> which considered vertical reference coding out to plus or minus twenty-six pels. Modified READ only uses vertical reference coding out to plus or minus three pels.

In both codes if a run does not end with a vertical reference code, the horizontal mode (HM) code precedes two Modified

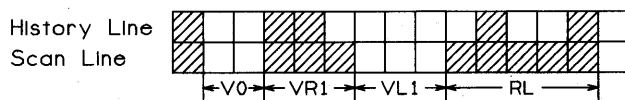


Figure 3—Coding examples.

Huffman run length codes for that run and the next run. Coding a pair of runs with one prefix is another way of taking advantage of the high probability that a run length code is immediately followed by another run length code.

Sometimes a run on the preceding line comes from an object which terminated on that line. A pass mode (PM) skips over such runs on the preceding line. It is used if another transition is encountered to the right of the first vertical reference transition (before the end of the run). A new run is started at that transition. Figure 4 illustrates pass mode. Pass mode can be coded several times within a single run. Table I gives the entropy per pel with pass mode present where the black and white runs have been considered separately. Table II shows how the distribution of runs shifts slightly when pass mode is used. Some runs are coded as a pass followed by a vertical code instead of as a run length.

In order to limit the effect of transmission errors, at least every  $K$ th line is coded one-dimensionally. The value of  $K$  is set as 2 for the normal resolution and 4 for the higher resolution.

The code table is given below.

Modified READ Code Table	
HM	001
PM	0001
V0	1
VL1	010
VR1	011
VL2	000010
VR2	000011
VL3	0000010
VR3	0000011
EOL	00000000001T
Extensions	
2-D	0000001xxx
1-D	00000001xxx

$T$  on the EOL denotes a tag bit which tells whether the next line is coded one-dimensionally ( $T=1$ ) or two-dimensionally ( $T=0$ ).

UNCOMPRESSED MODE

One of the future extensions has been identified as uncompressed mode. The xxx bits are '111' for this mode. This mode can improve compression significantly for scanned halftones and images with many one or two pel runs. IBM proposed the following table as a simple uncompressed mode coding scheme.<sup>17</sup>

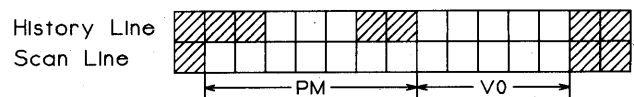


Figure 4—Pass mode followed by V0.

TABLE III.—Compression Ratios for High Resolution Images (1728 pels × 2376 lines)

DOC.	K=1		K=4 (EOLs)		K=2376 (no EOLs)	
	1-D STD	IBM	MR	IBM	MR	
1	13.7	19.3	19.8	26.3	28.4	
2	14.9	24.3	26.2	41.6	47.5	
3	7.9	12.4	12.6	17.6	17.9	
4	4.7	6.2	6.3	7.4	7.4	
5	7.5	11.4	11.6	15.5	15.9	
6	10.0	17.5	18.2	28.9	30.8	
7	4.8	6.2	6.3	7.3	7.4	
8	8.2	14.2	15.6	24.5	27.1	
AVE.	7.7	11.3	11.6	15.1	15.6	

Uncompressed Mode Code Table

Image Pattern	Code Word
1	1
01	01
001	001
0001	0001
00001	00001
00000	000001
Exits	
	0000001T
0	00000001T
00	000000001T
000	0000000001T
0000	00000000001T

The exit codes signal when to resume normal coding. The T denotes a tag bit which tells the color of the next run (Black = 1, White = 0).

TABLE IV.—Compression Ratios for Standard Resolution Images (1728 pels × 1188 lines)

DOC.	K=1		K=2 (EOLs)		K=1188 (no EOLs)	
	1-D STD	IBM	MR	IBM	MR	
1	13.7	15.7	15.7	21.2	21.4	
2	14.9	18.6	19.2	31.7	34.3	
3	7.9	9.9	9.9	15.0	14.8	
4	4.7	5.2	5.0	5.9	5.6	
5	7.5	9.1	9.1	12.7	12.6	
6	10.0	13.4	13.6	24.4	25.7	
7	4.8	5.2	5.1	5.9	5.7	
8	8.2	10.8	11.2	19.5	20.4	
AVE.	7.7	9.1	9.0	12.2	12.0	

COMPRESSION RESULTS

Eight documents of the line drawing and text type have been used to compare and evaluate the performance of coding schemes in CCITT Study Group XIV. Tables III and IV show the compression ratios for the CCITT one-dimensional standard (1-D STD) for each document at both resolutions. The compression ratios for the IBM scheme with error recovery (K=2 and 4) and for continuous two-dimensional coding without EOLs (K=1188 and 2376) are listed. The CCITT two-dimensional (MR) code compression ratios are also given.

ACKNOWLEDGMENT

The development of the IBM coding scheme was done in collaboration with Dr. G. Goertzel.

REFERENCES

1. CCITT Study Group XIV, "Report of the Meeting Held in Geneva, 14-18 November 1977," Doc. 25-E, 33 (1977).
2. CCITT Study Group XIV—Temporary Document No. 39-E, Kyoto 7-15 November 1979.
3. Huang, T. S., "Coding of Two-Tone Images," *IEEE Trans. on Comm.* COM-25, 1406 (1977).
4. Gorog, I., Heyman, P. M., Kim, H. K., and Lippman, M. D., "An Experimental Low-cost Graphic Information Distribution Terminal," *SID Dig. Tech. Papers*, 2, 14 (1971).
5. Huffman, D. A., "A Method for the Construction of Minimum-Redundancy Codes," *Proc. IRE* 40, 1098 (1952).
6. Shannon, C. E., "A Mathematical Theory of Communication," *Bell Syst. Tech. J.*, 27, 379 (1948).
7. Capon, J., "A Probabilistic Model for Runlength Coding of Pictures," *IRE Trans. on Inf. Theo.*, IT-5, 157 (1959).
8. Takagi, M. and Tsuda, T., "A Highly Efficient Run-Length Coding Scheme for Facsimile Transmission," *Electron Commun Jpn* 58, 30 (1975).
9. Deutsch, S., "A Note on Some Statistics Concerning Typewritten or Printed Material," *IRE Trans. on Information Theory* IT-3, 147 (1957).
10. Laemmel, A. E., "Coding Processes for Bandwidth Reduction in Picture Transmission," Report R-248-51, PIB-187, Poly Inst of Brooklyn, N.Y., 29 (1951).
11. Bahl, L. B., Barnea, D. I., and Kobayashi, H., Image Compaction System, U.S. Patent 3,833,900 (1974).
12. Yamazaki, Y., Wakahara, Y., and Teramura, H., "Digital Facsimile Equipment 'Quick-FAX' Using a New Redundancy Reduction Technique," *NTC '76*, 6.2-1 (1976).
13. Musmann, H. G. and Preuss, D., "Comparison of Redundancy Reducing Codes for Facsimile Transmission of Documents," *IEEE Trans. on Comm.*, COM-25, 1425 (1977).
14. Mitchell, J. L. and Goertzel, G., "Two-Dimensional Facsimile Coding Scheme," *ICC '79*, 8.7.1 (1979).
15. CCITT Study Group XIV, IBM Europe, "Proposal for Two-Dimensional Coding Scheme," Doc. 64-E, (1979).
16. CCITT Study Group XIV, Japan, "Proposal for Draft Recommendation of Two-Dimensional Coding Scheme," Doc. 42-E, (1978).
17. CCITT Study Group XIV, IBM Europe, "Uncompressed Mode Enhancement to IBM Coding Scheme," Doc. 80-E, (1979).

# Description and evaluation of a system for high-speed, three-dimensional computed tomography of the body: the dynamic spatial reconstructor\*

by RICHARD A. ROBB and BARRY K. GILBERT

*Biodynamics Research Unit, Mayo Foundation  
Rochester, Minnesota*

## INTRODUCTION

High temporal resolution, full three-dimensional imaging of the heart and circulation is required for accurate basic physiological studies of the structural-to-functional relationships of these organ systems, and for improved diagnostic evaluation and treatment of patients with cardiac and/or circulatory disorders. Current diagnostic imaging techniques and systems, including x-ray computed tomography (CT) (1,2), do not provide accurate measurements of the true dynamic changes in shape and dimensions of the intact heart over its entire anatomic extent, or for accurate measurement of the regional distribution of blood flow to, from, and within the heart, or any other organ of the body.

The approach to these problems which we are developing is that of high temporal resolution synchronous scanning of large cylindrical volumes of the body, using multiple x-ray sources and video fluoroscopic imaging techniques. A dynamic spatial reconstruction system (the DSR) has been designed and implemented to provide the temporally and spatially coherent multiple cross sections required to obtain the full three-dimensional anatomic and simultaneous hemodynamic information necessary for detailed quantitative analyses of regional function in any organ of the body, particularly the heart, and for more accurate diagnostic assessment of the localization, extent, and nature of disease and pathology which affects the circulation.

The anticipated performance and promise of the DSR have been extensively evaluated in simulation studies and by construction and application of a prototype system. Even though the DSR is based on both the engineering and mathematical principles of computed tomographic scanning, it is designed to achieve *dynamic volume* scanning, in contrast to static cross-sectional scanning, and therefore the DSR presents some unique problems by comparison to conventional CT systems and techniques. It is the purpose of this paper to address these problems and to describe approaches to their solutions. This will be done by first providing a description

of the DSR system, followed by discussion of potential DSR problems and results from DSR performance evaluation studies.

## *Description of DSR system*

The overall DSR system consists of much more than just the scanner. Figure 1 is a diagram of the overall DSR system, including associated data recording, processing, computation, and display facilities.

The DSR scanner has been extensively described (3,4,5). Briefly, it consists of 28 x-ray tubes equally spaced in a semicircle and 28 sets of image-intensifiers and video cameras arranged in the opposing semicircle behind a curved fluorescent screen. The entire gantry of 28 x-ray sources and associated imaging chains rotates continuously about the patient, at 15 revolutions per minute ( $1.5^\circ$  every  $\frac{1}{60}$  second), providing a new set of 28 views every  $\frac{1}{60}$  second and up to 240 equispaced views around  $360^\circ$  in less than two seconds. Up to 240 images of adjacent transverse sections less than 1-mm-thick are produced of a cylindrical scan volume up to 38 cm in transaxial diameter and 22 cm in axial height. One complete synchronous volume scan is accomplished in 0.01 second by pulsing each of the 28 x-ray sources in succession for 0.34 milliseconds each. The x-ray exposure for a 28 view scan is approximately 17 milliroentgens. These scans can be repeated 60 times per second.

To obtain high temporal resolution images (60/sec), successive sets of 28 views can be used for reconstruction. To obtain high spatial ( $\sim 1$  mm) and high density ( $\sim 1$  percent) resolution images of non-moving structures, up to 240 independent views can be used in the reconstruction. Desired tradeoffs between temporal, spatial, and density resolution can be achieved by selecting appropriate subsets of the total projection data recorded. This capability will permit detailed evaluation of the important relationships between temporal, spatial, and contrast resolution in computed tomography of the body, particularly relative to moving organs such as the heart, lungs, and circulation. The total incident x-ray dose for DSR scans is comparable to and in many cases less than

\* This work was supported in part by grants HL-04664 and RR-00007 from the National Institute of Health.



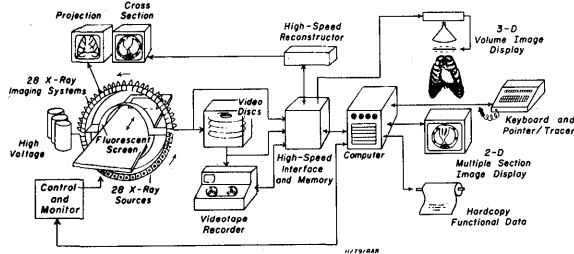


Figure 1—Major components and data flow paths associated with DSR system. (reproduced with permission from Robb et al. (10)).

the exposure received from clinically routine diagnostic x-ray procedures (3).

The DSR projection data is recorded on four video disc systems and transferred to the computer via a specially designed high-speed computer interface—the HSI (6). Computation of the data is accomplished by a combination of computer and high-speed parallel processing reconstruction techniques (i.e., algorithms implemented in special-purpose hardware). The hardware reconstructor, presently being fabricated (7), will permit computation speeds of about 10 milliseconds per cross section, or about 2.4 seconds for each volume comprised of 240 cross sections. Such rapid processing will be essential for effective practical use of the full potentialities of the DSR.

Off-line multi-dimensional display and analysis of the dynamic 3-D image data produced by the DSR scanner and computer will also be necessary. In addition to multi-oriented section displays and 3-D graphic representations of reconstructed volumes generated by the computer (8), a true 3-D display (illustrated at the right of Figure 1) is being developed, based on the vibrating mirror principle (9).

The computer system configuration for the DSR, previously described (10), is based on a ModComp CLASSIC computer, and is interfaced to the HSI and a Floating Point Systems array processor. This system is dedicated to control of the DSR scanner and to three-dimensional reconstruction and display of the high volume of images produced by the DSR. It is fully operational and ready to provide computational support for DSR-based projects.

#### *Evaluation of DSR performance using mathematical simulations and a prototype DSR system*

The performance of the DSR has been evaluated using a number of mathematical models and realistic computer simulations of its scanning geometry, timing, and photon statistics, and by extensive actual measurements of the physical characteristics and operation of a prototype assembly of one complete imaging chain of the DSR (x-ray tube, screen, image intensifier, relay lenses, and video camera). Such approaches are necessary since the capability and versatility of the DSR for permitting different programmed scan procedures in order to maximize either temporal, spatial, or contrast resolution or to optimize the tradeoffs among these

parameters, depending on the objectives of the study, cannot be tested until the DSR itself is fully operational.

The DSR scan procedures vary from conventional CT scans in several ways. First, a *cone beam of x-rays* is used in the DSR to image a relatively large axial volume of the body by producing a two-dimensional projection image on a fluorescent screen. In conventional CT systems, a pencil beam or fan beam of x-rays is used to produce a one-dimensional projection of a single transaxial slice through the body. Therefore the image reconstruction algorithm used for the DSR must account for divergence of the rays in the axial direction as well as in the transaxial direction. Secondly, when in the highest temporal resolution mode of scanning (60 volumes/sec), the DSR scan geometry produces three forms of *limited projection data* which are not encountered in conventional slow (2-20 second or longer) CT scans. These are: 1) relatively small number of views (28); 2) limited range of view ( $162^\circ$ ); and 3) incomplete field of view (21.4 cm transaxial diameter). Finally, demonstration of the capability for producing accurate *dynamic volume CT images* with the DSR is critically important.

This section describes mathematical simulation studies and testing of an actual prototype system which have either fostered direct solutions or initial approaches to these problems, or have demonstrated that their effect is not significantly deleterious to the hoped-for results. In many cases the simulation studies were validated using actual x-ray projection data obtained from the prototype DSR imaging system. This prototype system of the DSR is called the SSDSR (for Single Source Dynamic Spatial Reconstructor) and has been described previously (11,12).

#### **Cone beam vs. fan beam studies**

A cone beam of x-rays is used in the DSR to produce multiplanar (i.e., two-dimensional) x-ray video projection images for reconstruction of the *volume* of the body interposed in the x-ray beam. Although several investigators are working on development of true cone beam algorithms (13,14), no analytic closed form algorithm has yet been theoretically derived for reconstruction using a conical x-ray beam. As a result, current commercially available x-ray CT scanners must collimate the source beam to a fan and forgo the opportunity to collect projection data simultaneously for many adjacent cross sections. However, another approach is possible—assume the conical beam to be approximated by a “stack” of fan beams in the axial direction.

In the DSR, the relatively large x-ray source-to-object distance results in a small subtended angle in forming the projection image. The x-ray sources are 203 cm from the imaging screen and 145 cm from the center of the object to be reconstructed. This produces a magnification of 1.4, meaning that a cylindrical volume of the body 21.4 cm in diameter and 21.4 cm high is projected onto a 30 cm  $\times$  30 cm region of the screen. The maximum divergence of the rays above and below the central slice through this cylindrical volume is only  $4^\circ$ . That is, the divergence is greatest for transaxial slices through the volume at 10.7 cm above or below the central slice; the divergence is less for planes

nearer the central slice. Therefore, the 240 adjacent horizontal video lines which comprise the projection image produced by the DSR at any angle of view represent x-ray projections of *near-parallel* sets of fan beams, especially within a  $\pm 8$  cm region about the center, so that useful reconstructions over the entire anatomic extent of organs like the heart may be possible using conventional fan beam reconstruction algorithms applied separately to the data obtained from each adjacent video line in the projection image.

The reconstruction artifacts resulting from approximating a cone beam x-ray source by a stack of fan beams are presently undergoing investigation using three-dimensional mathematically generated non-axisymmetric phantoms of the human thorax, in conjunction with special computer programs which simulate both the passage of a conically shaped beam of x-rays through the phantom, and their absorption by a two-dimensional array of x-ray detectors (15). The resulting computer-generated cone beam x-ray projection data can be reconstructed with various reconstruction algorithms, followed by comparison with cross sections of the mathematical phantom at the same anatomic levels. The actual geometric and beam divergence characteristics of the DSR have been employed in these experiments and evaluated at the DSR computer facility (16).

The results of such a study are depicted in Figure 2. The left and right columns of images are cross sections through the mathematical phantom and their reconstruction by a fan beam algorithm (17), respectively, above and through the central plane of rays in the conical beam. Within the central plane of rays, the fan beam reconstruction algorithm yields geometrically and theoretically correct reconstructed cross sections. It is theoretically incorrect to reconstruct projection data generated via a conical beam of radiation using a fan beam reconstruction algorithm. Nonetheless, a comparison of the fan beam reconstruction with the section through the phantom at the same axial distance from the central plane of rays indicates that the fan reconstruction and the section through the phantom are quite similar. The assumption of adjacent fan beams may not, therefore, introduce major errors into reconstructions from the DSR cone beam data, due to a conical beam divergence in the axial direction of only  $4^\circ$ .

To further test this possibility, the SSDSR was arranged exactly in DSR geometry and used to produce real cone beam x-ray projections of the thorax of an experimental animal. The left panel in Figure 3 is an x-ray video projection of a dog's thorax recorded at one angle of view during rotation in the SSDSR, and depicts 64 brightened video lines, at different anatomic locations, spanning most of the volume of the thorax. These lines were selected from a total of 240 lines for reconstruction of 64 cross sections over the axial extent of the dog's chest. The video lines just above and below each brightened level were also digitized and averaged with the selected lines to produce cross sections effectively 3 mm thick, each 3 mm apart.

The right panel in Figure 3 shows these 64 reconstructed cross sections of the thorax. Each cross section contains  $64 \times 64$  picture elements and was reconstructed from 60

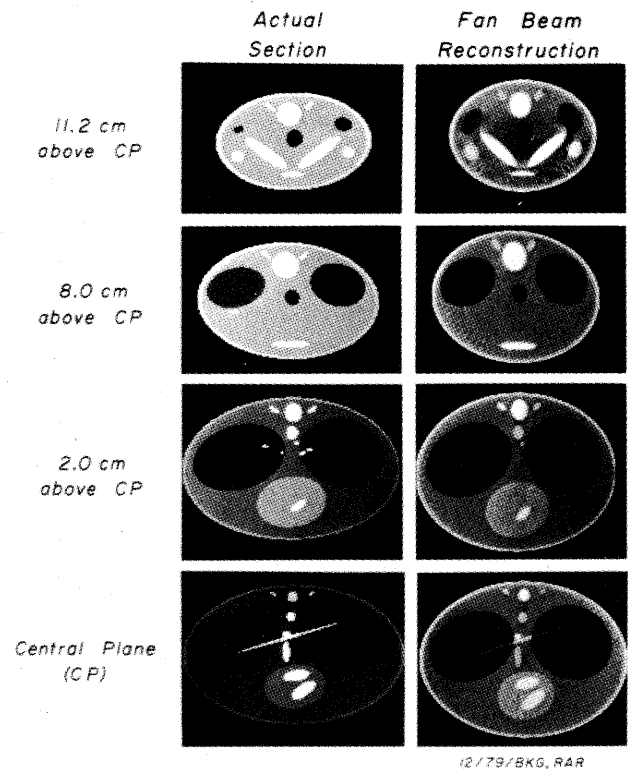


Figure 2—(Left column) Actual cross sections through three-dimensional non-axisymmetric mathematical phantom of human thorax at various levels above central plane of x-rays. (Right column) Fan beam reconstructions from 120 views around  $360^\circ$  of same sections shown at right. Each section is generated on a  $127 \times 127$  pixel grid; each pixel is  $.28 \times .28$  cm. Reconstructions are not plotted to exact scale of actual sections. "X-ray projections" were generated from the 3-D phantom via computer program which simulates polyenergetic conical x-ray beam. Some image artifacts are visible in upper right reconstruction, especially in left and right lung regions (dark regions near chest wall), resulting from fan beam reconstruction of cone beam projection data at extreme axial range of the cone beam. Other reconstructed levels are quite accurate by comparison to actual image, indicating potential usefulness of fan beam algorithm for cone beam data generated by DSR over  $\pm 8$  cm axial volume (reproduced with permission from Gilbert et al., (16)).

equally spaced views over  $360^\circ$  at a total x-ray exposure of 36 mr. The spinal vertebrae, ribs as they course through the plane of the cross section, the esophagus and bifurcated airway, the epicardial surface of the heart, the pleural surfaces of the lungs, and the diaphragm can all be visualized at the appropriate levels. Some trapped air can be seen in the transverse colon in the lower cross sections. No x-ray contrast material was injected into the dog to obtain these reconstructions, demonstrating that the photon statistics obtained with this fluoroscopic-based CT system are adequate for accurate 3-D reconstructions of the contents of the thorax.

Figure 4 is a computer-generated display of 56 separate, parallel 3-mm-thick coronal sections through the thorax, each 3 mm apart extending in the anterior-posterior direction from the sternum (upper left) to the backbone (lower right). Note the sections containing the cardiac catheters, the trachea, the esophagus, and the spinal column. These 3-mm-thick coronal sections of  $64 \times 64$  picture elements were com-

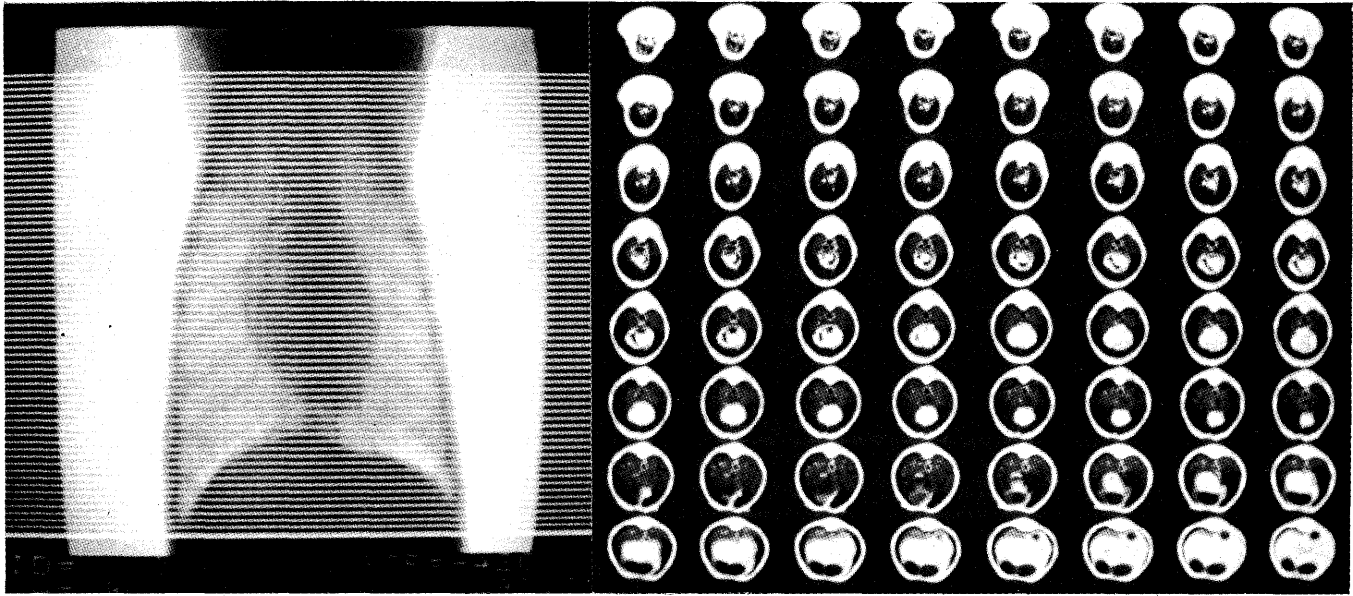


Figure 3—(Left) X-ray video image of canine thorax recorded by SSDSR system with brightened lines at 64 anatomic levels selected for reconstruction of 64 cross sections of dog's chest. (Right) Sixty-four reconstructed cross

sections of the thorax, each 3 mm thick, extending cephalocaudally from apex of lungs, at the upper left, to base of lungs, at lower right (reproduced with permission from Robb et al., (5)).

puted from the 64 reconstructed transverse sections of the chest shown in Figure 3. Sagittal and oblique sections could be similarly computed and displayed.

The multiple transaxial, axial, and oblique sections which can be simultaneously obtained from a *single* cone beam synchronous volume DSR scan provides the significant capability for "mathematically sectioning or slicing" any one or all regions of the scanned volume in any desired direction,

and for "zooming in" on any area of particular interest for detailed quantitative analysis. This capability might be called "non-invasive computerized dissection," similar to the capability of the surgeon or gross pathologist for examination and characterization of body tissues. The important difference, of course, is that the computerized scan, data processing, and display is non-destructive and painless, and therefore could be performed on unanesthetized patients.

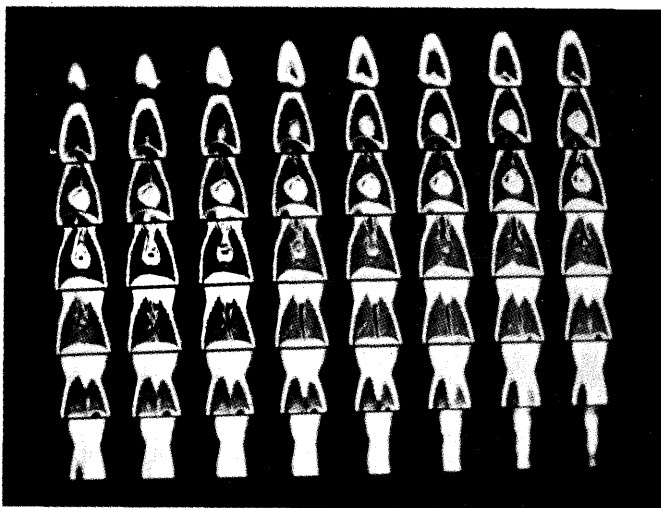


Figure 4—Computer-generated display of 56 separate, 3-mm-thick coronal sections through the thorax, extending from the sternum at the upper left to the backbone at the lower right, computed from 64 simultaneously scanned transverse sections shown in Figure 3 (reproduced with permission from Robb et al. (5)).

#### Limited projection data

In its highest temporal resolution mode of scanning (60 volumes/sec), the DSR will produce only 28 views over a limited 162° range. Furthermore, it will image a cylindrical volume only 21.4 cm in diameter and 21.4 cm high, so that objects larger than this size (most adult chests, for example) will exceed the field of view. If lower temporal resolution is acceptable, then all of these problems are eliminated because of the continuous projection data recorded during rotation of the DSR about the body, recording up to 240 views over 360° in 2 seconds. Transaxial diameters greater than 21.4 cm can be obtained by appropriately combining adjacent views. However, for imaging the heart, it may often be necessary to "stop-action" image at  $\frac{1}{30}$  to  $\frac{1}{15}$  sec, resulting in both a limited number of views over a range less than 180°, and a limited field of view in the projection data to be used for dynamic reconstruction of the volume imaged. These two problems (i.e., limited number/range of views and limited field of view) have been studied by simulations and SSDSR tests, and by applying techniques which reduce the artifacts which they cause.

## a) Limited number and range of views

Figure 5 shows four cross sections of a simulated human thorax reconstructed from mathematical projection data generated from a numerical model of the chest using DSR geometry and photon statistics and a conventional fan beam reconstruction algorithm (17). The model includes three simulated circular lesions in the right lung, 2 cm, 1 cm, and 0.5 cm in diameter, respectively, and radiopaque contrast material in the heart chamber and a coronary artery. The reconstructed images contain  $128 \times 128$  elements and were performed from 14, 28, 56, and 112 views over ranges of  $156^\circ$ ,  $162^\circ$ ,  $165^\circ$ , and  $166.5^\circ$ , respectively, to show the improvement in accuracy with increasing number of views. Further increases in the number of views will produce less and less improvement in accuracy once the number of views is approximately the same as the number of pixels on a side in the reconstruction image (18). Although the 112 view reconstruction exhibits the best resolution, the 28 view reconstruction is also geometrically accurate in spite of the heavy ray-shaped artifacts. The slight artifact in the chest wall at the bottom of each cross section is due to the limited range of views, but does not present a significant problem for study of the position and motion of the intrathoracic contents.

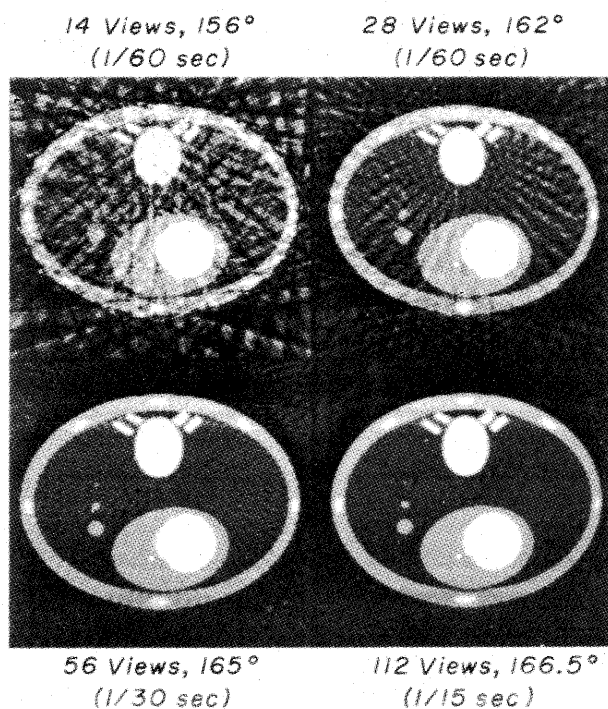


Figure 5—Reconstructed cross sections of simulated human thorax using DSR geometry and photon statistics. Reconstructions were performed using progressively increasing number of views, range of views, and scan times, as would be accomplished in programmed DSR scan sequences as the gantry rotates about the patient. Up to 240 views over  $360^\circ$  could be obtained in less than 2 seconds to further improve spatial and contrast resolution, if motion is not significant (reproduced with permission from Robb et al. (10)).

Since the DSR takes  $\frac{1}{15}$  second to collect 112 views, the temporal resolution in such images would not be as good as the 28 view images determined from  $\frac{1}{60}$  second scans. However, for imaging the lungs, this is an acceptable compromise since relatively little movement would occur in  $\frac{1}{15}$  second during breath-holding. And as the 112 view image suggests, tumors smaller than 5 mm in diameter could be detected from such DSR scans. Although motion would be detected better in 28 view scans, spatial resolution of only 5 or 6 mm is possible. No motion was assumed to have occurred in this simulation; therefore, the heart borders, the 2 mm diameter coronary artery, and chamber borders (note the 4 mm wide serrations in the chamber border) also are quite accurately reconstructed in the  $\frac{1}{15}$  sec, 112 view image. In a living patient, however, motion blurring would tend to degrade this accuracy, especially during the contractile and rapid filling phases of the heart cycle, so that  $\frac{1}{60}$  sec, 28 view scans may be necessary to capture the dynamics of the heartbeat during the rapid phases of the cardiac cycle.

Figure 6 is similar to Figure 5 but the reconstructions were performed from actual x-ray projection data of a dead dog's thorax recorded with the SSDSR. Again exact DSR geometry and typical DSR radiation levels were used. The image reconstructed from 28 views over  $162^\circ$  is geometrically accurate, but does contain density artifacts. In spite of the limited  $166.5^\circ$  range of view, the 112 view reconstruction exhibits both good spatial and good density resolution. Note the catheters and chambers of the heart, the bifurcated air-

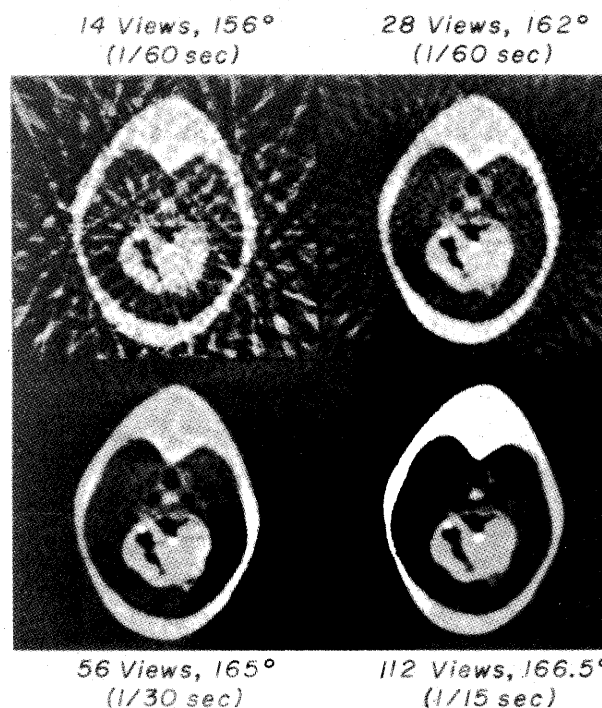


Figure 6—Four cross sections of dead dog's thorax reconstructed from different sets of actual x-ray projection data recorded with the SSDSR to simulate some of the same scan sequences which could be provided by the DSR (reproduced with permission from Robb et al. (10)).

way, esophagus, and boney detail of the spinal vertebrae. The DSR would be expected to produce images even better than these, since it is technologically superior to the SSDSR system in terms of geometric alignment and photon detection efficiency.

#### b) Limited field of view

If "truncated" projection images resulting from a limited field of view are used for reconstruction with conventional CT algorithms, serious artifacts are produced. These artifacts appear in the image in two general ways: 1) the regions of the body which are not included in every projection are geometrically distorted and a high density ring artifact appears at the boundary of the region of the body contained in every projection, and 2) the region of the body which is contained in every projection, although geometrically accurate in the reconstruction, is inaccurate in density values.

Figure 7 shows a simulation study of the effects of incomplete field of view on reconstruction of the thorax. The upper left image of Figure 7 is the actual cross section of a simulated 36 cm lateral diameter human thorax, with x-ray contrast material in a heart chamber. The upper right image simulates a DSR reconstruction using 28 views over 162°, assuming that the complete profiles of the entire chest at every angle of view are available. However, the bottom left panel shows the reconstruction that would actually be obtained if the DSR scan were performed in the high temporal resolution (60/sec) mode to freeze heart motion, since only the central 21.4 cm of the chest would be contained in every angle of view (i.e.,

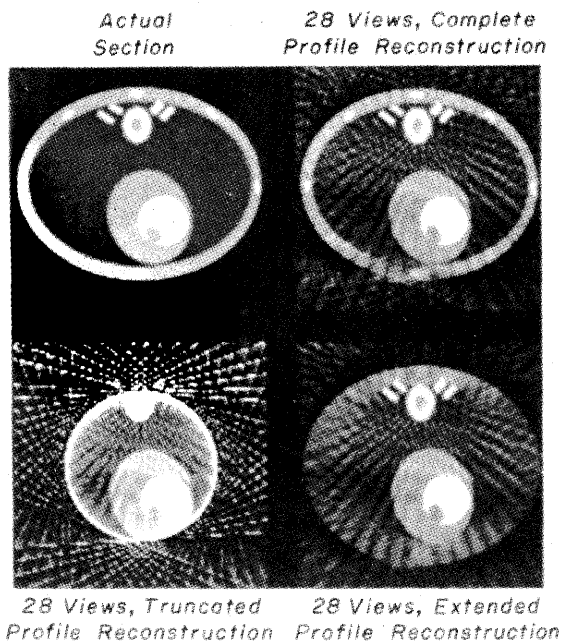


Figure 7—Comparison of reconstructions from truncated and mathematically extended profile data of mathematically simulated 36 cm diameter thorax using DSR geometry and high temporal resolution scans (60/sec) with limited 21.4 cm field of view (reproduced with permission from Robb et al. (10)).

about 40 percent of the total profile data is lost, or 20 percent on each end). The right-lower image is the reconstruction obtained by mathematically extending all truncated profile data back to 36 cm, using a technique developed by Lewitt (19) to extrapolate each truncated profile to zero in a smooth way consistent with the data to estimate the missing ends of the profile. Note the improvement in accuracy of density values in the heart and lungs (even though there are ray-shaped artifacts due to using only 28 views in the reconstruction); and note particularly the improvement in standard detail of the spinal vertebrae.

Figure 8 illustrates the effect of using this technique on actual truncated x-ray projection data. At the left of Figure 8 is a cross section of a dead dog's thorax, reconstructed from 120 views over 360° recorded with the SSDSR. If 20 percent of the entire number of samples in each profile are eliminated from each end of the profile (40 percent of total data omitted), then the reconstruction appears as in the center panel. However, if the truncated projection data is extended by the correction algorithm (19) (not by the actual samples!) prior to reconstruction, then the image appears as in the right-hand panel of Figure 8. The heavy ring artifact is noticeably absent, the densities of the heart and lungs are more accurate, and some structures outside of the field of view appear in the reconstruction, particularly the spinal vertebrae and the apex of the heart. Therefore, this technique promises to be very helpful in improving reconstructions from DSR scans in the high temporal resolution mode of scanning, where profiles may be truncated by 20 percent to 40 percent of their actual complete extent, depending on the size of the body scanned. In lower temporal resolution modes, the entire profile width will be available.

#### Dynamic CT imaging

To evaluate the capability of the DSR to dynamically image cross sections of the heart 60 times per second, a realistic mathematical model and numerical simulation of the thorax with dynamic geometric and densitometric variations in the heart muscle and chambers during the cardiac cycle was developed. The model was generated from detailed considerations and knowledge of myocardial mechanics obtained from indirect (e.g., EKG) and direct (angiography) measurements of actual hearts (8).

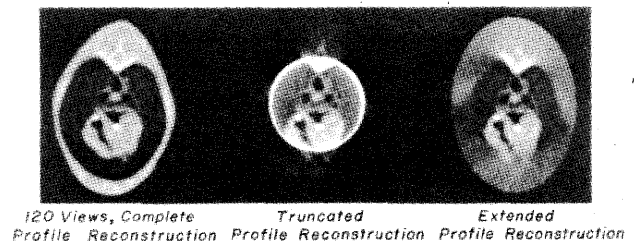


Figure 8—Cross section of dead dog's thorax reconstructed from 120 x-ray profiles recorded with the SSDSR, showing improvement in spatial and contrast detail if truncated projection data is mathematically extended before reconstruction (reproduced with permission from Robb et al. (10)).

Sixty images were simulated at  $\frac{1}{60}$  second intervals for a duration of 1 second (1 cardiac cycle), and projection data of the simulated thorax was mathematically generated for each of these images based on DSR geometry and scan timing. That is, 28 x-ray fan beam projections equally spaced over  $162^\circ$  were generated in the computer for each of the 60 simulated images of the thorax, with each successive set of 28 views incremented by  $1.5^\circ$  to simulate rotation of the DSR during each  $\frac{1}{60}$  second time interval. Reconstructions were then performed from these projection data to study the potential ability of the DSR to provide stop-action, high-repetition-rate imaging of cross sections of the heart.

Figure 9 shows reconstructions from these data compared to the actual simulated cross sections at four different time points in the simulated cardiac cycle. The reconstructed images have not been smoothed, scaled, or otherwise processed. They represent the actual output of the reconstruction algorithm. The reconstructions in the second and third rows of Figure 9 contain  $115 \times 115$  picture elements (pixels) and were performed using only 28 views over  $162^\circ$ , the same amount of data to be provided by the DSR in its highest temporal resolution mode of scanning. This is much less data than normally used in current conventional CT scanners (20); nonetheless, the geometry of the actual model is accurately reproduced in the reconstructions and no motion blurring is apparent. Such geometric accuracy suggests that important indices of myocardial function such as changes in wall thickness, muscle mass, and chamber volume could be accurately measured and studied. The accuracy of densities in the reconstruction is not as good, due primarily to the limited num-

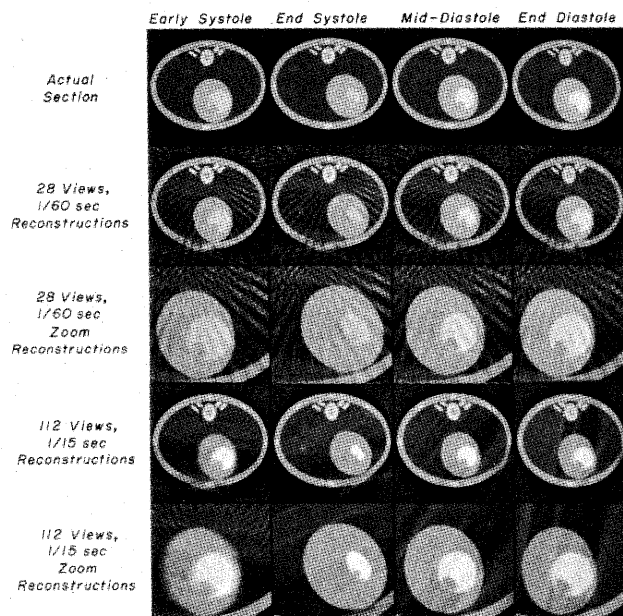


Figure 9—Actual section (top row) of simulated thorax with beating heart filling with x-ray contrast material at four different points in cardiac cycle compared with regular and “zoom” reconstructions (bottom four rows) from simulated DSR scan data for these same phases of the cardiac cycle, using 28 views and 112 views collected over  $\frac{1}{60}$  second and  $\frac{1}{15}$  second, respectively.

ber of views, as characterized by the relatively heavy ray-shaped artifacts emanating from the spine. However, even though the absolute values of the densities are incorrect, the relative changes in the heart are constant, suggesting that certain indices of hemodynamic function such as appearance and washout times of contrast material and relative myocardial perfusion ratios could be accurately studied. Dynamic motion pictures of these reconstructions in comparison to the actual model provide further evidence of the potential accuracy and usefulness of the DSR for dynamic CT imaging.

The third row of reconstructions in Figure 9 illustrates the results of a reconstruction technique which is advantageous for cardiac studies with the DSR, namely, performing a “zoom” reconstruction which includes only the region of the chest containing the heart. Such reconstructions can be obtained with a filtered back projection algorithm by back projecting all profile samples only over a selected “target” region (e.g., the heart) within the total scanned image (e.g., the thorax), and provide greater spatial resolution without increased computational cost by comparison to reconstruction of the entire region scanned if the same number of elements is used in the reconstruction matrix. Each reconstructed image in the third row of Figure 9 contains  $115 \times 115$  pixels, the same as in the second row, and was determined from the same projection data as used for the reconstructions in the second row. However, the back-projection portion of the algorithm (17) used to produce the reconstructions in the second row was performed only over the central one-half of the projection image region, resulting in approximately a two fold increase in spatial resolution.

Appropriately combining sets of 28 views together is another important reconstruction technique which can be used with the DSR for studying the tradeoffs between spatial and temporal resolution. As the DSR rotates at 15 rpm about the patient, it records 28 new views of the heart every  $\frac{1}{60}$  second from successive  $1.5^\circ$  increments of the gantry around  $360^\circ$ . Any selected set or combination of sets of these 28 views can be retrospectively selected to reconstruct the heart with varying temporal and spatial resolution. As shown in rows two and three of Figure 9, 28 view sets can be used to get the highest temporal resolution of 60/second. However, since the 28 x-ray sources are  $6^\circ$  apart on a  $162^\circ$  arc, 112 different angles of view are collected over an arc of  $166.5^\circ$  in a total of  $\frac{1}{15}$  sec before some x-ray source positions are repeated. Reconstructions obtained from these 112 views may provide greater spatial resolution than 28 view reconstructions, if significant motion does not occur during the  $\frac{1}{15}$  second interval required to record the 112 views.

The fourth and fifth rows in Figure 9 illustrate such 112 view,  $\frac{1}{15}$  second time interval reconstructions. The left-hand image was reconstructed during the early systolic (ejection of blood from chamber) phase of the heart cycle, where significant motion has occurred. This image is poor in quality compared to the image which was reconstructed from the same part of the cardiac cycle but from only 28 views recorded in  $\frac{1}{60}$  second. However, the 112 view reconstructions at the other points in the cardiac cycle where less mo-

tion has occurred exhibit improved quality by comparison to the corresponding images reconstructed from only 28 views at the same points in the cardiac cycle. The coronary arteries can be readily seen, particularly in the zoom images on the bottom row. These variations in image quality reflect the tradeoffs between temporal and spatial resolution where motion is either significant or relatively small during the  $\frac{1}{3}$  second scan interval.

The left-hand panel of Figure 10 is an x-ray video projection of a beating heart in an intact living dog recorded with the SSDSR at one instant in the cardiac cycle. The brightened horizontal line depicts one of approximately 80 levels of the heart, extending from base to apex, which could be selected for cross-sectional reconstruction of the heart during the cardiac cycle. During incremental rotation of the dog in the SSDSR through  $6^\circ$  on every other heart beat, which was controlled by the computer via a pacing electrode placed in the right ventricle, respiration was suspended and x-ray contrast material was continuously infused into the left ventricular chamber to achieve continuous beat-to-beat opacification of the chamber during the rotational scan period. The right-hand panel in Figure 10 shows eight different cross-sectional levels between the base and apex of the heart which were reconstructed at eight successive points throughout the cardiac cycle, with each successive point in time separated by 50 msec. The 6th level from the top corresponds to the

level indicated by the brightened line superimposed on the projection image at left.

The reconstructions in Figure 10 were performed using the zoom technique from only 28 views recorded over  $162^\circ$ , similar to the amount of data to be obtained from the DSR in its highest temporal resolution mode of scanning. However, it was necessary to use gated scanning (i.e., a new view is recorded on each successive heart beat, each of which is assumed to be exactly reproduced) since the SSDSR does not have the high-speed imaging capabilities of the DSR. Not every view included the entire thorax, so the correction technique for incomplete projection data (19) was used in the reconstruction process. The left ventricular chamber and epicardial surfaces of the heart can be seen in each successive section during the heart beat beginning with the contractile phases of the systolic period and ending with the subsequent diastolic filling period. A catheter can be seen in the first two levels (basal levels) and opacified coronary arteries can be readily detected in the first four levels. The change in shape and size of the left ventricular chamber is particularly graphic in the lower-most four levels (apical levels).

These images demonstrate the potential capability for and value (21) of obtaining the dynamic cross-sectional shape and dimensions of both the epi- and endocardial surfaces of the intact working heart over the entire anatomic and tem-

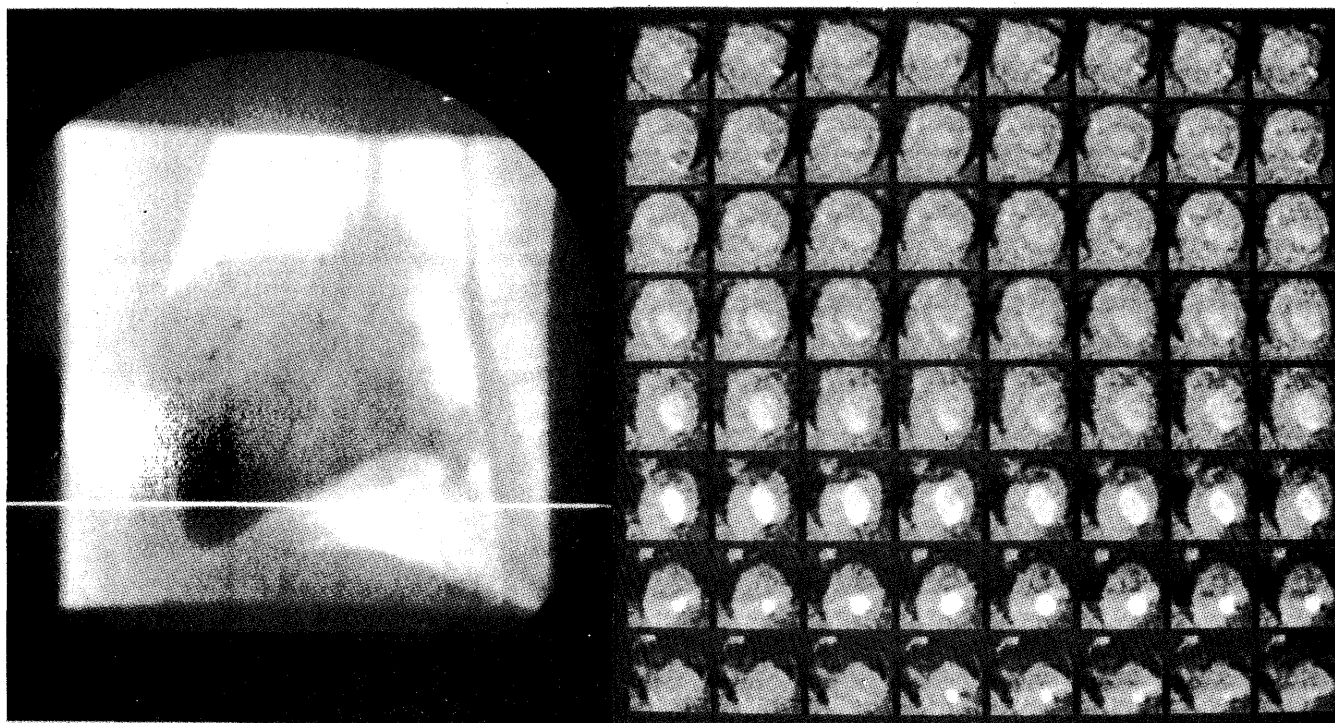


Figure 10—(Left panel) X-ray video projection of live dog thorax recorded with SSDSR at one angle of view during computer-controlled rotation, cardiac pacing, and infusion of contrast material into left ventricular chamber. (Right panel) Eight reconstructed levels (rows) at 8 time points (columns) throughout

the heart beat, beginning in space and time at the base of the heart (top row) and at the onset of systole (left column), and proceeding to the apex of the heart (bottom row) and end diastole (right column), respectively (reproduced with permission from Robb et al. (10)).

poral extents of the myocardial walls and cardiac cycles, respectively; that is, dynamic volume imaging.

## SUMMARY

High temporal resolution, full three-dimensional imaging of the body, particularly the heart, lungs and circulation, is required for accurate basic physiological studies of the structural-to-functional relationships of body organ systems, and for improved diagnostic evaluation and treatment of patients afflicted with disorders of the vital life processes for which these organ systems are responsible.

The SSDSR and DSR are the first known CT scanners to provide simultaneous scanning of up to 240 cross sections, that is, synchronous volume scanning. Consequently, the full three-dimensional extent of relatively large organs such as the heart can be captured in one scan procedure and stored in computer memory. The reconstructed 3-D image can then be mathematically (i.e., non-destructively) sectioned at any angle and displayed as multi-oriented slices through the anatomic volume. New radiographic images (reprojections) of the total volume scanned, or any desired subregion of the total volume scanned, can be formed without additional x-ray exposure, and displayed from any orientation with superimposed structures removed. Shaded surface displays of selected organs and true 3-D volume displays which permit operator interaction are also possible.

Many of the problems anticipated with the DSR have been thoroughly studied and evaluated using realistic simulations and actual data from a prototype DSR system—the SSDSR. These studies indicate that successful approaches to problems of cone beam geometry, low photon statistics, and limited projection data are possible. Techniques are also being developed in both software and hardware for easing the computational load and for facilitating comprehensive display of the data. The prospects for true dynamic volume imaging have been extensively evaluated with the SSDSR, which has demonstrated the powerful two new dimensions that the DSR will bring to computed tomography of the body—high temporal resolution and synchronous volume scanning. That is, the DSR will provide true stop-action ( $\frac{1}{100}$  sec) full three-dimensional imaging at a high repetition rate (60/sec) to make possible heretofore impossible basic investigative and clinical studies of organ systems and their processes throughout the body (22).

## ACKNOWLEDGMENTS

The authors wish to acknowledge their colleagues in the Biodynamics Research Unit at the Mayo Clinic and their collaborators upon whose multidisciplinary talents and efforts these studies have been largely based. Particularly important contributions came from Drs. E. L. Ritman and L. D. Harris at Mayo and from Drs. Gabor T. Herman and Robert M. Lewitt at SUNY/Buffalo. Thanks are extended to those who helped prepare this manuscript, Marge C. Fynbo and Marge A. Engesser for typing and artwork, Leo

O. Johnson for photography, and Mike J. Bozonie, Dennis P. Hanson, and Larry T. Thorson for programming assistance.

## REFERENCES

- Alfidi, R. J., MacIntyre, W. J., and Haager, J. R., "The effects of biological motion on CT resolution," *American Journal of Roentgenology*, 127:11-15, 1976.
- Boyd, D. P., Korobin, M. T., and Moss, A., "Engineering status of computerized tomographic scanning," *Optical Engineering*, 16(1):37-44, 1977.
- Ritman, E. L., Robb, R. A., Johnson, S. A., Chevalier, P. A., Gilbert, B. K., Greenleaf, J. F., Sturm, R. E., and Wood, E. H., "Quantitative imaging of the structure and function of the heart, lungs, and circulation," *Mayo Clinic Proceedings*, 53:3-11, 1978.
- Robb, R. A., Ritman, E. L., Gilbert, B. K., Kinsey, J. H., Harris, L. D., and Wood, E. H., "The DSR: A high-speed three-dimensional x-ray computed tomography system for Dynamic Spatial Reconstruction of the heart and circulation," *IEEE Transactions on Nuclear Science*, NS-26(2):2713-2717 (April) 1979.
- Robb, R. A. and Ritman, E. L., "High-speed synchronous volume computed tomography of the heart," *Radiology*, 133(3):655-661, 1979.
- Gilbert, B. K., Storma, M. T., Ballard, K. C., Hobrock, L. W., James, C. E., and Wood, E. H., "A programmable dynamic memory allocation system for input/output of digital data into standard computer memories at 40 megasample/s," *IEEE Transactions on Computers*, C-25(11):1101-1109, 1976.
- Gilbert, B. K., Chu, A., Atkins, D. E., Swartzlander, Jr., E. E., and Ritman, E. L., "Ultra high-speed transaxial image reconstruction of the heart, lungs, and circulation via numerical approximation methods and optimized processor architecture," *Computers and Biomedical Research*, 12:17-38, 1979.
- Harris, L. D., Robb, R. A., Yuen, T. S., and Ritman, E. L., "The display and visualization of 3-D reconstructed anatomic morphology: Experience with the thorax, heart, and coronary vasculature of dogs," *Journal of Computer Assisted Tomography*, 3(4):439-446, 1979.
- Traub, A. C., "A new three-dimensional display technique," Report #M68-4 of the MITRE Corporation, Bedford, MA, 1968.
- Robb, R. A., Lent, A. H., and Chu, A., "A computer-based system for high-speed three-dimensional imaging of the heart and circulation: Evaluation of performance by simulation and prototype," *Proceedings of the Thirteenth Hawaii International Conference on System Sciences*, 3:384-408, 1980.
- Robb, R. A., Greenleaf, J. F., Ritman, E. L., Johnson, S. A., Sjostrand, J. D., Herman, G. T., and Wood, E. H., "Three-dimensional visualization of the intact thorax and contents: A technique for cross-sectional reconstruction for multiplanar x-ray views," *Computers and Biomedical Research*, 7:395-419, 1974.
- Sturm, R. E., Ritman, E. L., Johnson, S. A., Wondrow, M. A., Erdman, D. I., and Wood, E. H., "Prototype of a single x-ray video imaging chain designed for high temporal resolution computerized tomography by means of an electronic scanning dynamic spatial reconstruction system," *Proceedings of the San Diego Biomedical Symposium*, 15:181-188, 1976.
- Minerbo, G. N., "Convolutional reconstruction from cone beam projection data," *IEEE Transactions on Nuclear Science*, NS-26(2):2682-2684 (April) 1979.
- Altschuler, M. D., et al., "Demonstration of a software package for the reconstruction of the dynamically changing structure of the human heart from cone beam x-ray projection," Technical Report No. MIP632, Department of Computer Science, SUNY/Buffalo.
- Altschuler, M. D., Chang, T., and Chu, A., "Rapid computer generation of three-dimensional phantoms and their cone beam x-ray projections," *SPIE Application of Optical Instrumentation in Medicine VII*, 173:287-290, 1979.
- Gilbert, B. K., Schwartau, W. K., Chu, A., Beistad, R. D., and Krueger, L. M., "Hardware implementation of arithmetically demanding image reconstruction algorithms for x-ray computed tomography," *IEEE Transactions on Pattern Analysis and Machine Intelligence* (in press).



17. Herman, G. T., Lakshminarayanan, A. V., Naparstek, A., Ritman, E. L., Robb, R. A., and Wood, E. H., "Rapid computerized tomography" in Laudet, M., J. Anderson, and S. Begon, *Medical Data Processing*, London, Taylor and Francis, Ltd., 1976, pp 581-598.
18. Shepp, L. A. and Logan, B. F., "The Fourier reconstruction of a head section," *IEEE Transactions on Nuclear Science*, NS-21(3):21-43 (June) 1974.
19. Lewitt, R. M., "Processing of incomplete measurement data in computed tomography," *Medical Physics*, 6(5):412-417, (October) 1979.
20. Boyd, D. P. "Status of diagnostic x-ray CT," *IEEE Transactions on Nuclear Science*, NS-26(2):2836-2839 (April) 1979.
21. Robb, R. A., Harris, L. D., and Ritman, E. L., "Computerized x-ray reconstruction tomography in stereometric analysis of cardiovascular dynamics," *Proceedings of the Society of Photo-Optical Instrumentation Engineers*, 89:69-82, 1976.
22. Wood, E. H., "New vistas for the study of structural and functional dynamics of the heart, lungs, and circulation by non-invasive numerical tomographic vivisection," *Circulation*, 56(4):506-520, 1977.

# 3-D Viewer for interpretation of multiple scan sections

by BRENT BAXTER

*University of Utah Medical Center  
Department of Radiology*

## 1. INTRODUCTION

A new viewing device is being constructed which will allow a physician to examine multiple scan sections simultaneously in their proper orientation in all three dimensions. Test images already produced on a laboratory prototype viewer clearly show parallax and stereoscopic depth cues characteristic of a real three dimensional object. Our experience viewing X-ray CT and ultrasound sections has shown that certain anatomical and pathological features important in cancer detection are more readily appreciated on the 3-D viewer than on conventional planar imaging facilities such as at T.V. monitor or film. The viewer can also form 3-D graphical figures with lines oriented from left-to-right, top-to-bottom and front-to-back. By moving one's head, distant lines appear to move relative to nearby lines, just as would be expected.

An exciting application of this apparatus is in the guidance of a specially constructed surgical instrument for brain biopsy of suspected tumors. Biopsy of small deep lesions can be a formidable procedure requiring prolonged anesthesia, multiple "blind" needle insertions with occasional serious complications from hemorrhage or loss of function. The probability of these undesirable consequences can be minimized by utilizing a special stereotaxic surgical instrument guided by CT image data seen on the viewing device. First, a CT scan is made of the head with the stereotaxic frame attached, in which both the suspected tumor and the frame can be seen. Next a simulated probe is placed at the center of the lesion and the probe track is positioned to avoid sensitive structures such as major blood vessels, etc. The biopsy procedure is then performed using numerical coordinates supplied by the computer for positioning the biopsy needle. The advantage of this new technique is that it makes it possible to reach deep lesions 2-3 mm in diameter without the trauma of a large bone incision and without the uncertain positioning of hand guided procedure. The combination of accurate needle guidance and the ability to operate through a small burr hole may make it possible in many cases to obtain the biopsy specimen under local rather than general anesthesia.

## 2. 3-D VIBRATING MIRROR DISPLAYS

In 1961 Muirhead pointed out that a thin aluminized film tightly stretched over a ring could be deformed pneumati-

cally to form a spherical mirror of reasonably high quality, and that the curvature, either concave or convex (and hence the focal length) could be controlled by changing the air pressure on the rear surface of the film. Sometime later Traub (2,3) showed how the variable focal length associated with a mirror of changing curvature could be used to construct a 3-D display that provides both stereopsis and parallax depth cues. The primary components of the system as it applies to viewing CT and ultrasound B-scan images are shown in Figure 1. Planar images, obtained in the usual way from conventional CT and ultrasound apparatus, are placed on the cathode ray tube in sequence beginning with the one at the front of the display volume and so on through the set, the entire sequence being repeated rapidly enough to avoid flicker (about 30-40 times per second). If one were to simply view the cathode ray tube directly, the superposition of the planar images would discard information in the depth direction obtained at such great cost by the CT scanner. The purpose of the vibrating mirror is to spread these individual images out in depth and thereby provide the observer with the same kind of depth impression he would have if he could look into a solid object that was both transparent and luminous. In this case, luminosity is a function of X-ray absorption (CT number). Spreading the images out in depth is accomplished by synchronizing the mirror vibrations with the planar image display schedule in such a way that the mirror is in its extreme convex position when the plane nearest the observer is being displayed, and as the mirror curvature changes, successive images appear to emanate from planes displayed in depth, behind the first. In this way, it is possible to construct a display volume whose apparent extent in the depth direction depends on the mirror's amplitude of vibration. If the process is repeated rapidly enough, persistence effects in human vision create the impression that all planar images appear continuously, each one at a different distance from the observer, even though each plane appears on the cathode ray tube screen for only a small fraction of the total cycle.

Line drawings are produced in a different manner, by means of a 3-D point plotting mechanism. Consider, for example, a single centrally located stationary spot of light on the CRT screen. As the mirror vibrates, this spot of light is spread out in depth creating the illusion of a line extending along the central axis of the display volume. Additional lines (of any arbitrary orientation) can be created from rows of

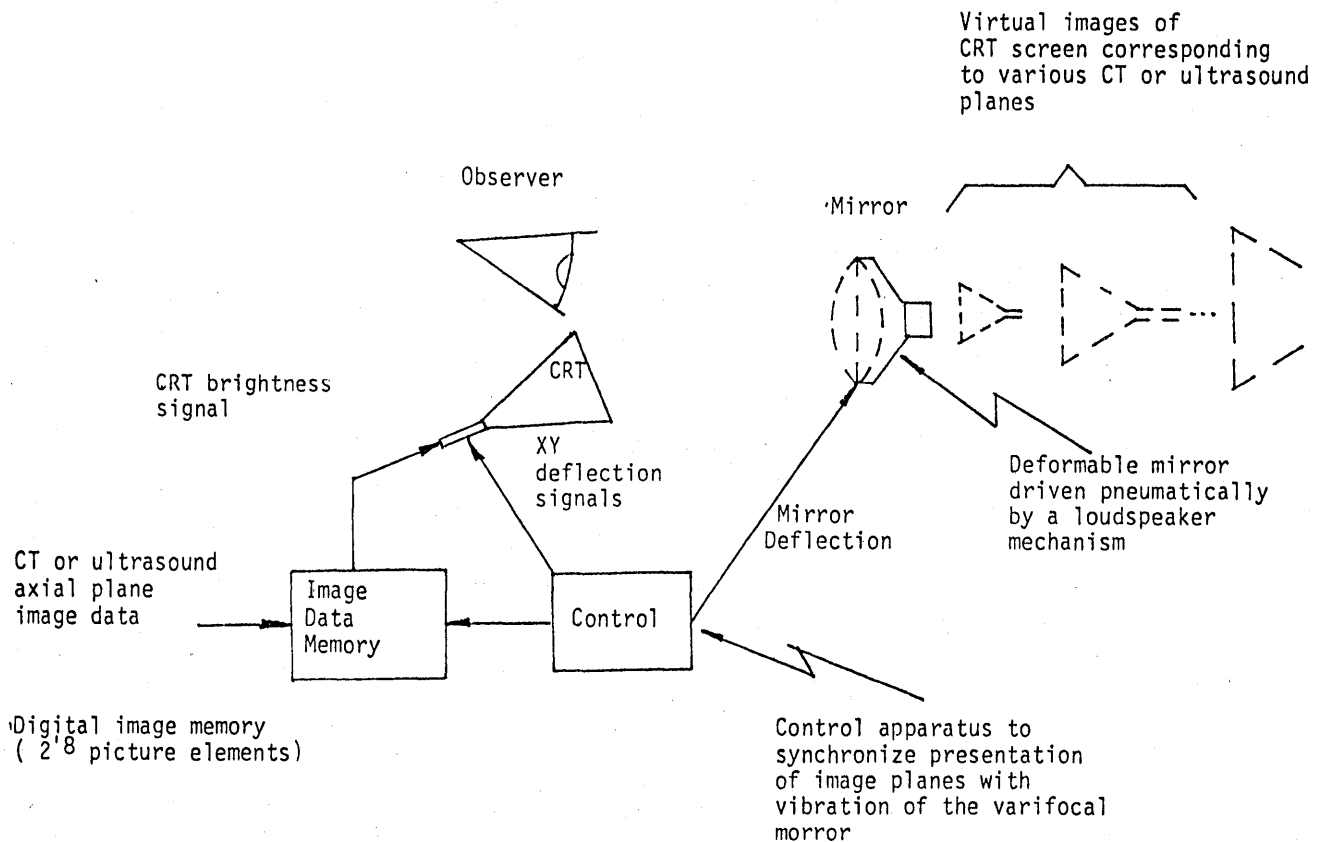


Figure 1—A three dimensional virtual image is formed in the space behind the deformable mirror as successive planes are displayed on the cathode ray tube screen. Line drawings may be produced by flashing a series of adjacent

spots on the CRT screen in synchronism with the mirror vibrations. Cycling through the data rapidly (30-40 times per second) creates the illusion of a stationary three dimensional virtual image.

abutting points by flashing spots of light at the proper position on the CRT screen at the proper instant in time. Lines produced in this way are not constrained to lie in one or more widely spaced planes, as described previously, but may be drawn anywhere within the display volume.

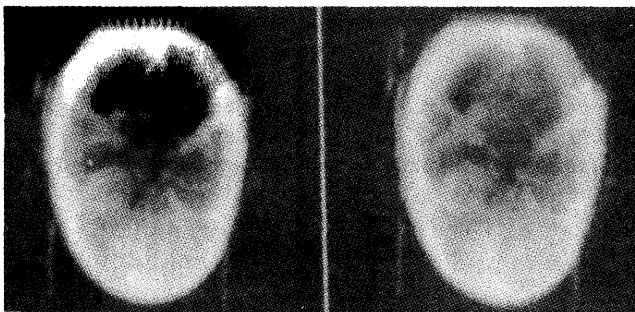


Figure 2—Stereo prints may be made by photographing an entire series of head scans displayed simultaneously on the 3-D viewer. The depth illusion produced by viewing this pair with a stereoscope simulates the effect seen by an observer from a single direction. Moving one's head while viewing the image on the 3-D viewer produces additional depth cues not available on this stereo pair.

### 3. EXPERIMENTAL RESULTS

Experiments conducted using a laboratory prototype viewer have provided several interesting findings:

a) The eye/brain system seems to fill in the space between virtual image sections producing an illusion of continuity in depth. A CT scan series showing a dilated biliary duct system appeared to be very realistic, with no clue to the fact that the 3-D image was constructed from 8 image planes separated by approximately 1-cm. CT scans of the head also appeared to be continuous in the depth direction except near the skull when the display was viewed from off axis, where it was possible to see the skull outlines on individual planes. This subjective continuity effect was observed on a variety of case material. Our CRT system has a short phosphor decay time (30 microseconds) suggesting that the visual system rather than the display apparatus is responsible. This unexpected finding is important because it reduces, and in many cases eliminates, the need for interpolation between virtual image planes. It is interesting to note that conventional planar (2-D) digital imaging devices frequently require interpolation facilities to prevent the appearance of a regular pattern of dots or lines superimposed on the image even

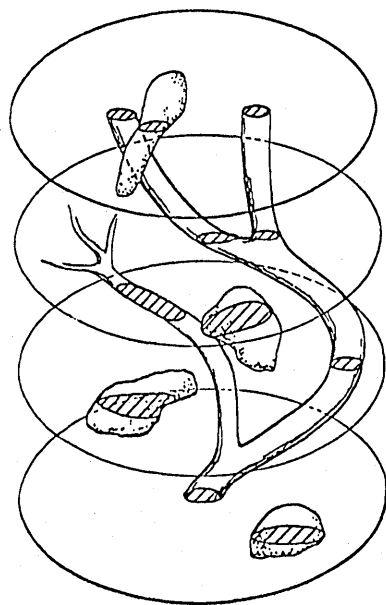


Figure 3-a

Figure 3—In (a) four CT planes are superimposed showing how the structure may be distinguished from isolated masses by taking advantage of correlations

when the grid points are closely spaced. TV raster lines are an example of this type of artifact. The requirement for interpolation is described in relation to nuclear medicine images by Baxter (4).

b) A second somewhat frustrating finding is the difficulty encountered in producing good photographic reproductions of the 3-D image for use in presentations. Unlike holograms, which are usually made of solid objects, the 3-D image is transparent, allowing an observer to see into the structure. Depth cues obtained by changing the vantage point are important for a realistic impressing of depth, yet we have experienced difficulty reproducing this impression photographically, with stereo film pairs (Figure 2) or in the form of a movie. A CT head scan movie was made with the camera in a fixed position, the mirror vibration being increased gradually, spreading the virtual images out in depth. Since the movie was made from an oblique angle, it is possible to see the gradual separation of skull outlines on successive planes with increasing mirror vibration amplitude. Even so, the depth effect is much more pronounced when the image is seen first hand. One possibility we have not yet investigated is the use of multiplex holograms as a means of reproducing the illusion of depth seen first hand on the viewer. Goodman (5) is investigating their use to display 3-D computer data.

c) 3-D Image Visualization: Figure 3 shows how the tree-like biliary system can be traced with the aid of correlated features on adjacent planes, thus distinguishing it from isolated potentially cancerous masses. In liver studies where contrast material has been injected into the biliary system, the tree-like appearance is very striking and is reminiscent

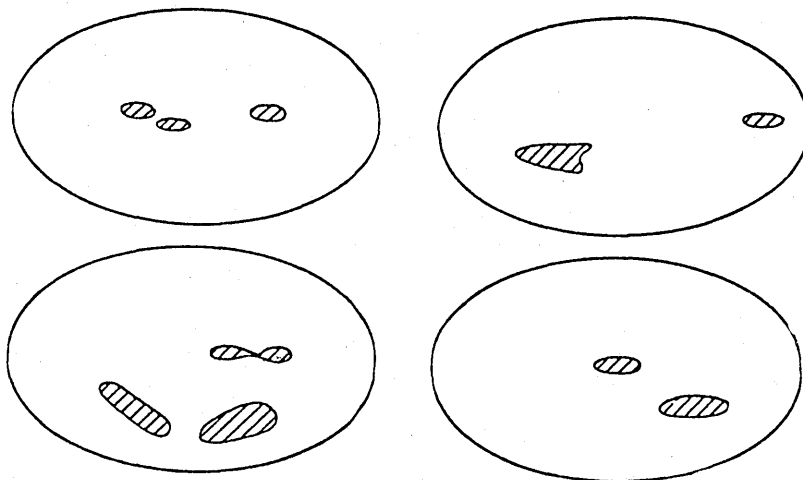


Figure 3-b

between adjacent planes. In (b) this same data is shown in a conventional planar format.

of an uprooted plant with the dirt shaken off. The same image data viewed section by section is much more difficult to visualize.

We expect that another important application of the viewer will be in examining complex abdominal structures where it is difficult to relate the various organs to one another from sectional images viewed separately.

#### 4. APPLICATION: BRAIN BIOPSY GUIDANCE

A special surgical frame (6,7) has been used successfully to reach small targets in a head phantom without the targets being seen. The current technique requires the neurosurgeon to select a target point on a CT section demonstrating the lesion and also an entry point on a separate CT section thus fixing the probe path and the biopsy site. The problem with using conventional TV viewing devices is the difficulty in visualizing the entire surgical field from separately viewed CT sections.

A pseudo 3-D line drawing computer graphics device\* has been used to display both anatomical image data and surgical frame outlines; however, it is necessary to extract contours representing bone, tumor, soft tissue structures, etc. before the device can be used effectively. This contour extraction step is subject to corruption by noise and it imposes an ad-

\* Evans and Sutherland Picture System II.

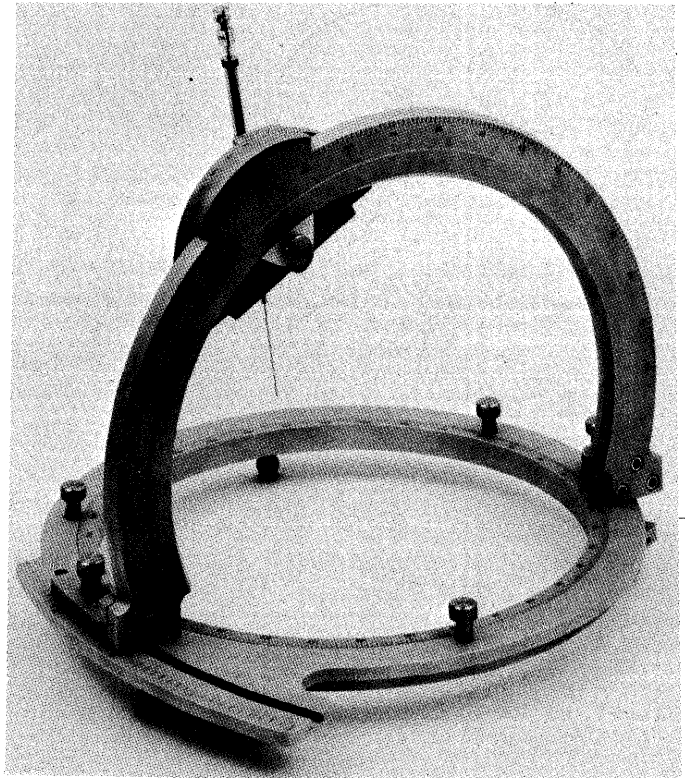
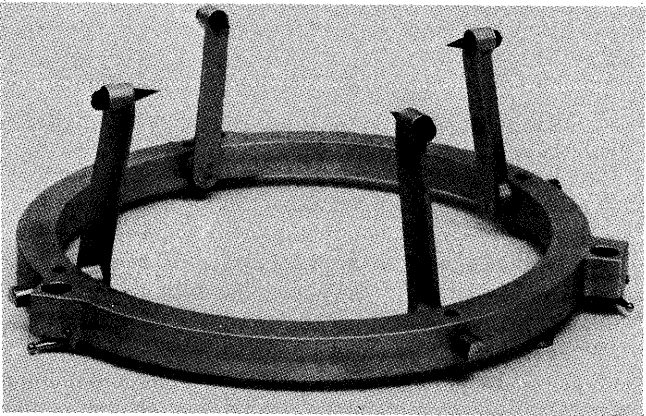
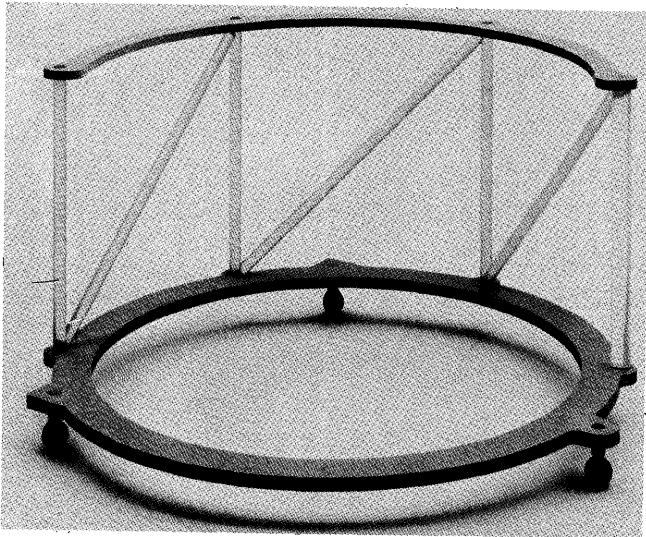


Figure 4—A prototype stereotaxic surgical frame of this type is attached to the head during CT scanning and probe placement, assuring a fixed spatial relationship between the CT image of the head and the frame. Vertical and diagonal localizer bars are used to determine the location of the frame relative to the CT image. A probe may be made to follow a manually placed simulated probe track visible on a computer graphics display. In use the surgeon will manipulate the simulated track with the resulting scale settings being supplied by the computer.

ditional computational burden. Also, motion parallax depth cues are absent.

The 3-D viewer overcomes these difficulties by providing the neurosurgeon with a view of the entire CT scan series upon which a stylized movable outline of his surgical instrument is superimposed. By moving the simulated probe into position of the 3-D image he can choose the path least likely to cause complications. The computer then supplies the numerical frame settings to be used in guiding the biopsy needle. Figure 4 shows how this instrument is constructed.

## 5. CONCLUSION

The anatomical realism afforded by the 3-D viewer will enable a physician to examine complex organ systems without the need to mentally superimpose multiple scan sections.

We expect this feature to be useful in planning and performing biopsy procedures in the brain for its greater reliability and reduced risk to the patient as compared with current methods.

## REFERENCES

1. Muirhead, J. C. *Rev. Sci. Inst.* 31: 210, 1961.
2. Traub, A. C., *Applied Optics* 6: 1085, 1967.
3. Traub, A. C., U. S. Patent #3,493,290.
4. Baxter, B. S., "Precision computer display techniques in nuclear medicine." *Proc. 23, International Symposium SPIE*, San Diego, August, 1979.
5. Goodman, J. W., "Personal Communication." Department of Electrical Engineering, Stanford University.
6. Brown, R. A., "A computerized tomography/computer graphics approach to stereotaxic localization," *J. Neurosurg* 50: 715-720, 1979.
7. Brown, R. A., "A head frame for use with CT body scanners," *Invest. Radiol* 14: 300-304, 1979.

# Absolute limits on image processing

by DAVID G. BROWN, ROBERT F. WAGNER, and MARY PASTEL ANDERSON

Medical Physics Branch, Division of Electronic Products  
Bureau of Radiological Health, FDA  
Rockville, Maryland

## INTRODUCTION

The metrology of image processing for medical diagnostic imaging systems can be developed from two different approaches. One method involves the comparison of actual clinical efficacy of one processing scheme versus another. A paradigm for this approach is the ROC analysis technique outlined by Metz<sup>1</sup> and by Swets.<sup>2</sup> The second method, taking as its starting point axioms of information theory proposed by Shannon,<sup>3</sup> calculates the performance of an ideal observer from the system imaging parameters for some idealized imaging task. Image processing can only improve the performance of the human observer up to that of the ideal observer, in the limit of "ideal" processing.

Consider the situation in computed tomography (CT) imaging for which the detection of low contrast "lesions" is often a problem of clinical interest. The series of images shown in Figure 1 show a CT scan of a plastic phantom with moderately low-contrast disc-shaped inserts. The image on the left is from a normal scan, that in the center simulated as though from a scan with a factor of 122 less dose, and that on the right a grossly smoothed version of the center image. Note that the larger disks are easily visualized (with a greater than two orders of magnitude savings in patient dose as dividend) but that the smaller disks are still hidden in the mid-frequency components of the noise. Thus, by being willing to give up information contained in the image but not required for the task at hand, one may use a processed image made with a much lower dose. The question being alluded to in this example is how to be able to tell, for a particular detection task and presuming ideal image processing, how low a dose could be used for a CT scan.

## THEORETICAL FORMULATION

The imaging system is regarded as a two component system, separated into detection and display stages. In the detection stage object information is encoded by the system. As explained in an earlier paper, information theory concepts can be applied to this process to yield an information transfer function for the system. In the conventional notation as developed by Shaw<sup>4</sup> and extended by the authors,<sup>5,6</sup> this is the spectral detective quantum efficiency (DQE( $f$ )).

$$DQE(f) = DQE(0) MTF_s^2(f) / MTF_N^2(f)$$

where DQE(0) is the quotient of the number of noise equivalent quanta apparent in the image divided by the actual number of quanta used in making the image,  $MTF_s(f)$  is the modulation transfer function of the total system, and  $MTF_N(f)$  is that portion of the MTF which acts upon the image noise.

In the case of computed tomography, for example, the noise will be acted upon by the reconstruction algorithm. Assuming that  $MTF_s$  is factorizable into algorithmic ( $MTF_{alg}$ ) and non-algorithmic ( $MTF_{na}$ ) components, one obtains

$$DQE(f) = DQE(0) MTF_{na}^2(f)$$

where  $MTF_{na}$  is due primarily to the finite focal spot size, detector aperture and other geometrical factors.

As the information transfer function of the system, DQE( $f$ ) describes the fidelity with which object information at each spatial frequency is encoded by the detection stage of the imaging system. Obviously, information not encoded cannot be successfully displayed no matter what processing is employed. Thus, DQE( $f$ ) represents one absolute limit on the potential for improvement in imaging performance by means of such processing.

Except for the trivial case for which information over some frequency domain is completely lost because  $MTF_{na}(f)$  is identically zero over that domain, an observer detection criterion is necessary for the calculation of limitations on observer performance, and our focus shifts to the display stage of the system. As formulated by Wagner and others<sup>5,7</sup> from decision theoretic considerations, a signal to noise ratio criterion is the natural choice. The observer is assumed to scan the image to determine the presence or absence of a known signal  $s$ . By "scan" is meant the convolution of a template (ideally  $s$  itself) with the image. Of course in the frequency domain the convolution will be a multiplication, resulting in considerable simplification. The ratio of peak signal to rms noise or display signal-to-noise ratio (SNR) may now be written

$$SNR = \frac{\iint S MTF_s MTF_o df_x df_y}{\left[ \iint N MTF_o^2 df_x df_y \right]^{1/2}}$$

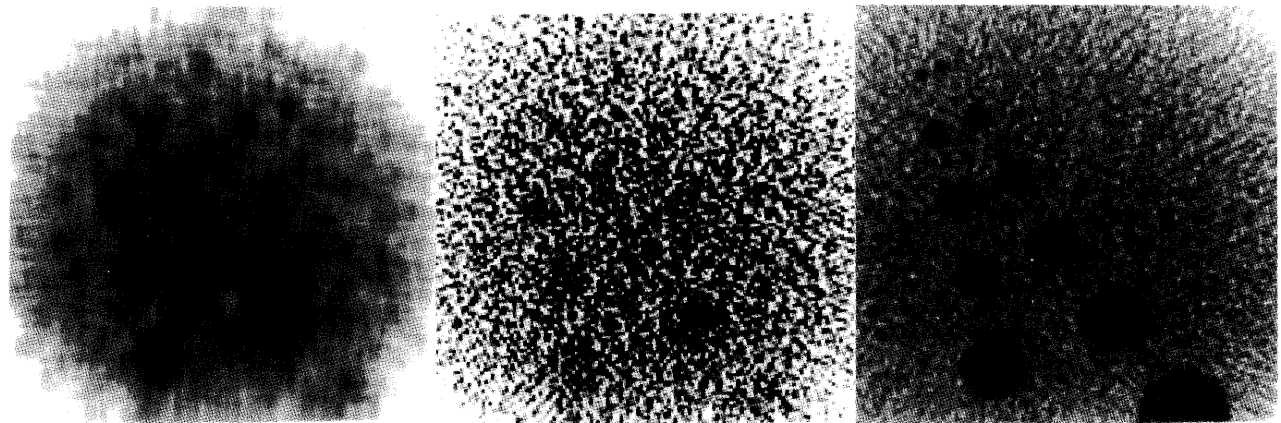


Figure 1—Low-contrast hole pattern as seen in (left) a normal CT scan, (center) a simulated low-dose CT scan, (right) a low pass filtered version of the center scan.

where  $S$  is the signal spectrum (Fourier transform of the object),  $N$  is the noise power spectrum,  $MTF_i$  is the MTF of the imaging system, and  $MTF_o$  is that of the observer.

This equation may be used to generate families of curves quantifying the performance of possible observers for any given imaging system and detection task. For example, if  $k$ , the signal-to-noise ratio in the image at the observer's de-

tection threshold is used for "SNR," if the system and observer MTF's and shape of the system noise power spectrum ( $N$ ) are specified, and if the dose is given (to normalize  $N$ ), then a relationship holding at detection threshold between two of the parameters characterizing the object being imaged may be established. Figure 2 gives just such a relationship for peak contrast and disc diameter for the detection of disc-

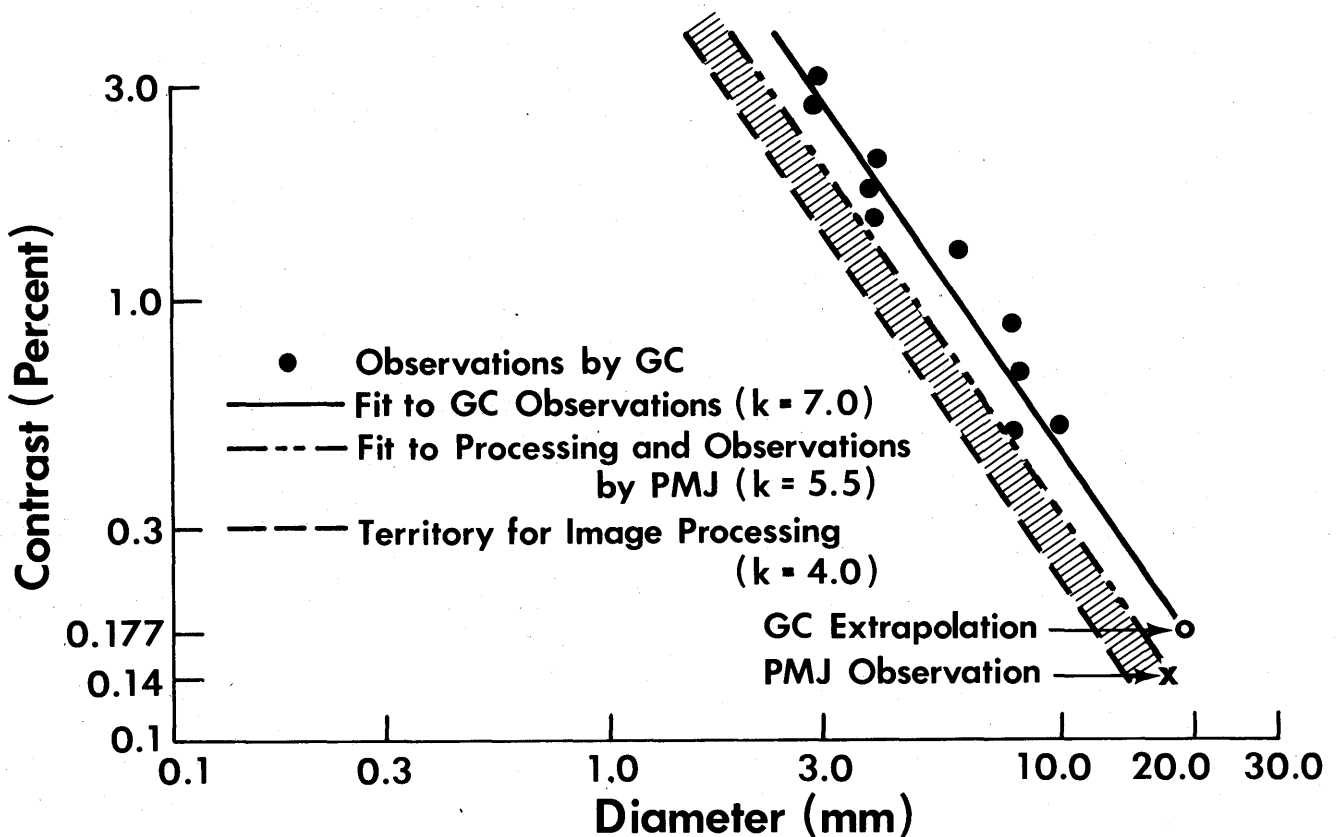


Figure 2—Contrast-detail-dose diagrams for an EMI Mark I CT system.

shaped objects—commonly referred to as a contrast-detail-dose (CDD) diagram.

It has been shown that the optimal observer scans with an aperture which prewhitens the noise and then matches the “prewhitened” blurred signal,<sup>5,8</sup>

$$\begin{aligned} \text{MTF}_o &= S \text{MTF}_s / N \\ \text{SNR} &= \left\{ \iint \frac{S^2 \text{MTF}_s^2}{N} dx dy \right\}^{1/2} \end{aligned} \quad (2)$$

The CDD's for this ideal observer are shown in Figure 2 for three different threshold signal-to-noise ratios. The system parameters used to generate the curves were obtained from measurements described in Reference 5. The right-most curve is fit to the experimental observations of Gerald Cohen,<sup>9</sup> yielding a  $k$  of 7. The middle curve is fit through observer performance data of Peter Joseph,<sup>10</sup> who used image processing to improve the detection of large, low contrast objects. Finally, the left-most curve is drawn for  $k$  equals 4, which is compatible with studies of optimal human observer performance.<sup>11</sup> Thus, image processing could be expected at most to result in the improvement of imaging performance by a shift from the  $k=7$  to  $k=4$  curves, a non-negligible but limited range of improvement.

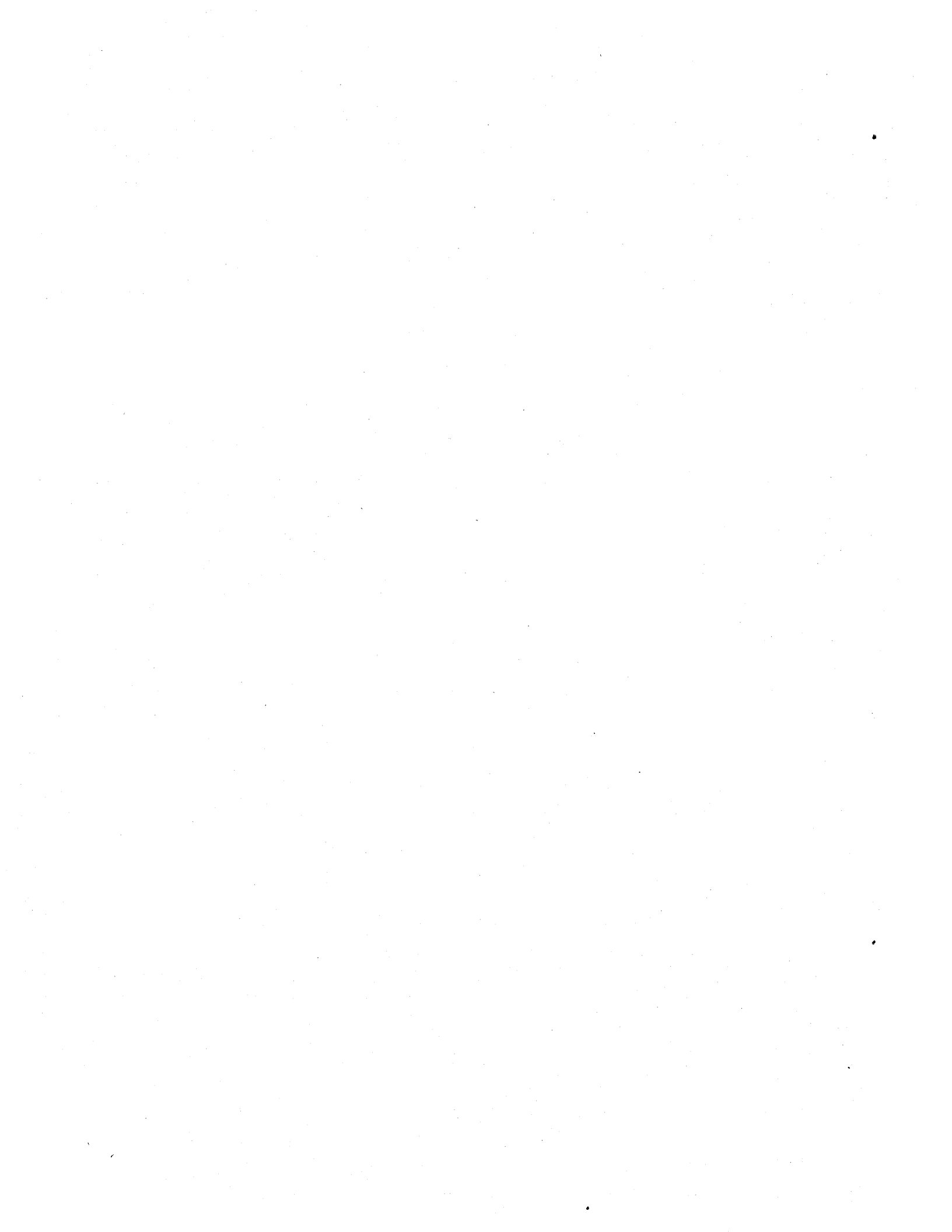
## SUMMARY

By calculating the performance of an ideal imaging system for a particular detection problem, a limit on the potential benefit to be derived from image processing may be derived. The CDD diagram formalism is one method for graphically illustrating such a limit. Since dose varies as the square of the signal-to-noise ratio, for CT we conclude that a dose reduction of on the order of  $(7/4)^2$  or about a factor of 3 is the gain which might be achieved through the use of optimal image processing.

## REFERENCES

1. Metz, C. E., *Seminars in Nuclear Medicine*, VIII, 283 (1978).
2. Swets, J. A., et al., *Science*, 205, 753 (24 August 1979).
3. Shannon, C. E., *Proc. IRE*, 37, 10 (1949).
4. Shaw, R., *Rep. Prog. Phys.*, 41, 1103 (1978).
5. Wagner, R. F., Brown, D. G., and Pastel, Mary S., *Medical Physics*, 6(2), 83 (1979).
6. Brown, D. G., Anderson, Mary Pastel, and Wagner, R. F., *Proc. SPIE*, (1979).
7. Wagner, R. F., *Photog. Sci. Eng.*, 22, 41 (1978).
8. Hanson, K. M., *Medical Physics*, 6(5), 441 (1979).
9. Cohen, G., *J. Comput. Assist. Tomogr.*, 3(2), 197 (1979).
10. Joseph, P. M., *Radiology*, (to be published).
11. Rose, A., in “Vision: Human and Electronic,” (New York: Plenum, 1973).





# Generalized methodology for the comparison of diagnostic imaging instrumentation

by LEON KAUFMAN and DALE SHOSA

UCSF Radiologic Imaging Laboratory  
So. San Francisco, California

## INTRODUCTION

Technology is providing exciting new ways to diagnose and characterize disease in a non-invasive manner. The decrease in morbidity and mortality that is realized by substituting pneumoencephalography and cerebral angiography by x-ray computed tomography (CT) and brain radionuclide imaging, or coronary angiography by radionuclide ventriculography and perfusion studies, cannot be easily quantitated by cost-effectiveness studies. The dollar value of peace of mind, patient dignity and comfort, avoidance of blindness or loss of limb, stroke, etc., cannot be derived from system analysis.

What some see as an unhealthy proliferation of diagnostic modalities has led to concern about excessive use. It can be argued that the marketplace in the form of medical history outcome, relative costs and patient acceptance, will eventually lead to convergent practice in the choice of diagnostic modalities. If such is the case, quantitative evaluation of instrumentation becomes redundant. In reality it serves many valuable purposes: a) at a minimum it provides a reasonable basis for quality control, b) it points the way toward an understanding of the diagnostic process and consequent possible improvements, c) it delineates the limits of a technique, and d) it provides the rationale for the exploration of improved techniques. This is only true if the instrument evaluation process is carried out in a competent manner. If not, it can have effects totally opposite to those listed above.

It is important to keep in mind that the evaluation of a new imaging technology in terms of the assumptions, parameters and procedures that have been found adequate for existing technology can yield misleading results. A case in point is presented by nuclear imaging, where evaluation has been based on considerations of instrument sensitivity and modulation transfer function (MTF) (mainly for scintillation cameras and scanners). For emission tomographic devices characterization has been in terms of sensitivity and spatial resolution (either point spread function, PSF, or line spread function, LSF). The use of high contrast objects such as bar patterns and point sources to measure contrast and/or resolution as well as the assumptions of uniform scatter contributions to the output image (1), can lead to evaluations of the instrument that are consistent only for the type of object being used for that evaluation: The effects on low

contrast lesions due to structured backgrounds introduced by complex sources and scattering material distributions, long tails in the PSF, and texture introduced by the detectors or processing algorithms, are not properly accounted for. Instead, photon detection efficiency becomes an over-weighted parameter. In fact, we have experimentally demonstrated that detector efficiency is not as important a parameter as expected from traditional image evaluation methods (2,3).

The growth in intradisciplinary imaging methods (i.e., planar imaging, single-photon emission tomography and positron tomography in nuclear medicine), as well as interdisciplinary modalities (nuclear, x-ray computed tomography (4), and nuclear magnetic resonance, NMR (5), imaging) both point toward the need for techniques that allow for a meaningful intradisciplinary comparison, and which can, conceptually, be extended to interdisciplinary comparisons as well.

In a previous communication (6) we have adopted the Rose model (7) to allow us to characterize an imaging instrument by an estimate of the number of elements,  $T$ , that can be resolved within an image. Thus, we write

$$T = \frac{NC^2}{k^2(1+C)} \quad (1)$$

where  $N$  is the number of discrete events forming the image,  $k$  is the confidence factor, and  $C$  is the contrast, defined by Rose as the signal-to-background ratio, with a value between 0 and 1. The factor  $(1+C)$  was added to take into account the effects of noisy signal as well as noisy background (6). For  $k=2$  and  $k=3$  we have a 97.7 and 99.8 percent confidence factor, respectively. Rose states that "the factor  $C^2$  is a consequence of the contrast  $C$  and the random character of photon distributions; the factor  $k^2$  reflects both the random character of the photon distribution and the need to avoid false alarms" (7).

Contrast is defined in Rose's formulation as "a measure of the signal as a function of the background brightness, that is,  $C = \Delta B/B$  and  $0 \leq C \leq 1$ " (7) where  $\Delta B$  is the difference in intensity between the site of the lesion and background  $B$ . In nuclear imaging one must sometimes consider the case  $\Delta B/B > 1$ .

Analytically a false alarm occurs when an element in a uniform background appears somewhat brighter or some-

what dimmer than its surroundings, where this difference is due only to statistical fluctuations. On the other hand, in diagnostic imaging the physician usually searches for patterns that involve more than one resolved element, or for differences in intensity in certain parts of the field. Thus, not every false alarm necessarily generates a false diagnosis. In many other procedures (for instance, cardiac wall motion studies), the delineation of an area is sought, and the exact contrast difference between adjacent elements is less significant. Based on these considerations, it would appear that, depending on the type of study being performed, satisfactory diagnostic images can be obtained with values of  $k$  between 2 and 3. The estimate is in need of corroboration through systematic study, but does not affect the methodology presented here.

Rose's formulation is particularly attractive in that it separates the statistics of the image (represented by  $N$  and  $k$ ) from the physics of the imaging process, which affects the value of  $C$  in the output image. In this manner, the tradeoffs between instrument sensitivity and the parameters that yield contrast can be easily evaluated. Rose's formulation also makes it explicit that contrast is independent of the number of counts in the image, the latter only affecting the certainty with which the contrast can be determined. Although very basic, and supported experimentally (8), this concept is sometimes overlooked in the evaluation of imaging instrumentation. In nuclear medicine, this misunderstanding has led to the concept that images are "statistics limited." As a consequence, undue emphasis has been placed on instrument efficiency, to the detriment of consideration of other important instrument characteristics which degrade performance (2). In x-ray computed tomography, this leads to the belief that increased dose by itself can improve image quality, where in fact it may only lead to better definition of artifacts produced by beam hardening, motion, detector instability, misalignments, etc.

The extension of Rose's model presented here is intended to provide a framework within which these pitfalls are avoided. Within the constraints that: a) it is understood that signal-to-noise (the basis for Rose's model) is a necessary but not sufficient criterion of visibility, and b) factors other than optimal signal-to-noise can be legitimately used to choose a particular diagnostic imaging modality, we demonstrate that meaningful quantitative comparisons of these modalities can be obtained.

## SUMMARY OF PRIOR WORK

Rather than to repeat here the detailed methodology used in expanding Rose's model (7), we summarize the main results. We considered the following sources of degradation of the resolving power of an imaging system:

### *Spatial resolution*

The finite area over which data from a point in the subject is distributed in the output image produces a loss of observed

contrast. For a spherical lesion and an approximately Gaussian point spread function (PSF), the output contrast  $C$  can be conveniently approximated (6) by the expression

$$C = C_o \exp(-FWHM^2/D^2) \quad (2)$$

where  $C_o$  is the object contrast,  $FWHM$  is the full width at half maximum of the Gaussian PSF and  $D$  is the lesion diameter. Equation 2 is valid only for  $FWHM/D < 1.5$  (6).

### *Texture*

If a uniform object is imaged, and the variability in resultant intensity with respect to the mean value is measured for regions in the image with area  $a$ , one of three types of distributions are generally found:

- i) Normal distribution in value and in space, the value being that which is expected on the basis of counting statistics only. This form of noise is the one treated by Rose.
- ii) Same distribution as in i above, but with a value that exceeds by a noise magnification  $M$  that which is expected from counting statistics. Noise of this kind is introduced by the reconstruction algorithms used in 3-D reconstruction (9).
- iii) Texture, where after correction for counting statistics a remnant  $\beta$  is found that is not normally distributed in value or in space. General consideration of this case is difficult, but it can be approximated as follows: In general the distribution of values for  $\beta$  can be approximated by a normal distribution, as we have demonstrated for the scintillation camera. In addition, even when  $\beta$  has a non-random distribution in space, the location of a lesion is randomly distributed with respect to background structure. Thus, for an ensemble of images, the effects of  $\beta$  can be approximated by those of a random distribution.

The effect of texture can be included in Rose's formulation in a number of ways (6). We have chosen to include it as an explicit alteration of output contrast of the form

$$C'^2 = C^2 - k^2\beta^2 \quad (3)$$

where  $\beta$  is the fractional value of the standard deviation of counts in a region of area  $a$  with respect to the counts in a surrounding (background) region, these corrected for pure counting statistics and  $M$ . Thus, in the presence of texture, acquired quanta can be thought of as being partitioned into two groups—one which produces increased detectability of small structures and the other which serves to enhance the visibility of interfering texture in the background. This is an important effect, and in many situations typical of diagnostic nuclear medicine, the images produced by scintillation cameras are as much "texture limited" as they are "statistics limited," and increases in count density with the accompanying increase in dose-time product will not yield commensurate increases in diagnostic efficacy.

### Other sources of alteration of contrast

Image contrast is altered by effects other than finite spatial resolution and texture.

In general, we can write

$$C = \frac{C_o FF}{1 + \Sigma f} \quad (4)$$

where  $\Sigma f$  is a sum of factors, examples of which are

- i)  $SF$ , the scatter fraction as defined in Reference 1, with one important difference: Rather than being constant over the image, it is used only as a local parameter, defined in the immediate vicinity of the lesion. It can be different even over the area of interest and the chosen background region (10).
- ii)  $P$ , the accidentals overlap rate (also locally defined).  $P$  is mostly important in coincidence imaging, but, as demonstrated by Lewellen (11), also affects scintillation camera images obtained at high rates.
- iii)  $E$ , the electronic noise normalized to real signal, a factor that is not important in photon-counting instruments, but can be significant in devices that obtain data by integration of large numbers of discrete events.
- iv)  $CR$ , the collimator crosstalk, which can be large in multi-peak and high-energy imaging, but is not trivial even under more favorable conditions (12).

$FF$  is the filling factor, introduced to take into account the loss of contrast produced by the finite dimensions being sampled in a direction perpendicular to the image plane. In a planar camera  $FF$  is the fraction of the signal that is generated by the target as opposed to that generated by target plus non-target tissues present in the projection (accounting, of course, for attenuation effects). In a tomographic camera  $FF$  is the ratio of lesion size (in the direction mentioned above) to the slice thickness.

Combining Equations 2 and 4, we can write, for a spherical lesion,

$$C = \frac{C_o FF \exp(-FWHM^2/D^2)}{1 + \Sigma f} \quad (5)$$

where the  $FWHM$  refers to the sharp part of the PSF, and all parameters are considered in their local context. It is important to note that the parameters  $SF$ ,  $P$ ,  $E$ ,  $CR$ , etc., are not generally independent of each other, as is often assumed.

## DERIVED PARAMETERS

### Effective spatial resolution

Equation 2 defines output contrast as a function of  $FWHM$  for a well behaved PSF (i.e., Gaussian or nearly Gaussian without appreciable tails).

If the sharp component of the PSF is accompanied by the presence of tails due to, for instance, the acceptance of scatter, or forced sharpening of resolution by image processing

or reconstruction algorithms, equivalence between output contrast and PSF  $FWHM$  is lost.

To obtain a realistic measure of spatial resolution we can define as an effective resolution the  $FWHM_{eff}$  that yields the measured contrast  $C$  for cylindrical or spherical lesions of known diameter  $D$  assuming that the system is well behaved. If the system is not well behaved  $FWHM_{eff}$  will not be identical to the  $FWHM$  obtained from a point source. Thus, from Equation 2, for a sphere,

$$FWHM_{eff} = D[\ln(C_o/C)]^{1/2} \quad (6a)$$

It can similarly be shown (6) that for a cylinder

$$FWHM_{eff} = .83D/[\ln(C_o/C_o - C)]^{1/2} \quad (6b)$$

The attractiveness of this definition of resolution is that it takes into account significant effects that are lost in the measurement of high contrast sources, it unmask manipulations of the image, and presents results within the context of relevant diagnostic situations. An unattractive feature is that no single number may characterize resolution, which now becomes object-dependent. Unfortunately, reality does not necessarily permit characterization of the instrument in terms of simple parameters.

### Needed sensitivity

For an instrument that obtains images by accumulating serially or in parallel a number of discrete events,  $N$  in Equation 1 can be expressed as

$$N = N_o St \quad (7)$$

where  $N_o$  is the rate with which discrete events are generated for the purpose of forming an image (disintegration rate in nuclear imaging, x-ray tube output in x-ray CT, resonant nuclei in NMR, etc.);  $S$  is the sensitivity of the system, given as the fraction of all events that is incorporated in the output image; and  $t$  is the imaging time. The value of  $S$  can be reduced by dead time effects, which need be taken into account.

For low contrast images, the sensitivity needed to obtain a certain resolving power under a given set of imaging conditions is

$$S_n = Tk^2 M^2 (1 + C) / N_o t (C^2 - k^2 \beta^2) \quad (8)$$

where the number of resolved elements is  $T = A/a$ ,  $A$  being the area of interest and  $a$  defining the size of lesion being sought. To the extent that the actual sensitivity  $S$  can be less than  $S_n$ , the instrument becomes inadequate for the particular imaging problem.

### Imaging efficiency

If we define  $S_p$  as the sensitivity that a perfect instrument ( $C = C_o =$  object contrast) would need under the same imaging conditions, the imaging efficiency (relative to that per-

fect instrument) is

$$S_p/S = (C^2 - k^2\beta^2)(1 + C_o)/C_o^2 M^2(1 + C) \quad (9)$$

This is an important parameter, since it allows us to obtain an understanding of how far an instrument is operating from its statistics-limited point for a particular imaging problem and to accurately appreciate the gains that can accrue from improvements in specific performance parameters.

### Resolving power

Under conditions of adequate area over which background can be estimated (6), Rose's formulation of resolving power can be modified to take into account texture as follows

$$T = N(C^2 - k^2\beta^2)/k^2(1 + C) \quad (10)$$

where for the moment we ignore  $M$ . Recalling once more that  $T = A/a$  and  $N$  is defined over the area  $a$ , Equation 10 can be expressed as

$$a = Ak^2(1 + C)/N(C^2 - k^2\beta^2) \quad (11)$$

The term  $A/N$  is the square of the normally distributed noise in the image except for texture, as defined before, measured for regions of area  $a$ . Thus, in a case where  $N$  and  $M$  are not easily obtained (as with CT scanners, ultrasound or NMR), we can write

$$a = k^2(\text{noise}^2)(1 + C)/(C^2 - k^2\beta^2) \quad (12)$$

keeping in mind that both the noise and  $\beta$  vary as a function of  $a$ . Note also that the parameter  $C$  takes into account the filling factor as well as any other sources of degradation of contrast.

Equation 12 allows us to estimate the resolving power of an instrument by choosing an area  $a$ , as test point, and observing whether the measured parameters in an image reduce to a result for Equation 12 such that  $a \leq a_r$ . In that case, the instrument can resolve lesions of area  $a$  with a confidence  $k$ . The resolving limit is, of course, the case where  $a = a_r$ . Alternatively, Equation 12 can be expressed in terms of  $k$ ,

$$k = C / \left[ \frac{(\text{noise}^2)(1 + C)}{a} + \beta^2 \right]^{1/2} \quad (13)$$

Thus, the output characteristics of the instrument are sufficient to compare its imaging performance to any other modality operating on the same subject. For infinite counts Equation 13 reduces to  $k_\infty = C/\beta$ .

## EXAMPLES OF COMPARISONS

### Intradisciplinary comparisons

#### Nuclear imaging, same radiopharmaceutical and same modality (i.e., planar) with different instruments

As an example of the results of this methodology we will compare a scintillation and high purity germanium camera (13) in two planar imaging conditions.

a. A lesion of 1cm diameter, with a 10:1 lesion to background tissue label, within a 19cm-diameter tissue sphere surrounded by a 0.5cm-wide shell with a 5:1 shell to background tissue label. The lesion is at a 5cm depth, and is imaged from a direction such that the distance between it and the collimator is at a minimum. We will assume that the background tissue contains  $0.1\mu\text{Ci}/\text{cm}^3$  of Tc-99m. This example attempts to represent a reasonable benchmark for brain tumor imaging. The product  $C_o FF$  is calculated to be 0.51. For the scintillation camera we will use the following parameters: With a high resolution collimator  $FWHM = 6.7\text{mm}$  (6),  $SF \sim 0.8$  (14),  $\beta = 0.025$  (6),  $M = 1$ ,  $CR = 0.06$  (12) and, at low rates both  $P$  and the dead-time will be assumed to be nil. The sensitivity, or product of detector efficiency times the collimator acceptance is  $4.8 \text{ cts/sec}/\mu\text{Ci}$  (13). The expected output contrast  $C$ , calculated from Equation 5, will be 0.175. For  $k=2$ , the factor  $(C^2 - k^2\beta^2) = 0.028$ .

For the HPGe camera with a high resolution collimator  $FWHM = 4\text{mm}$ ,  $SF \sim 0$ ,  $\beta = 0$ ,  $M = 1$ ,  $CR = 0.02$ , and the sensitivity is  $2.5 \text{ cts/sec}/\mu\text{Ci}$  (13). The resultant contrast will then be 0.426 and the term  $(C^2 - k^2\beta^2) = 0.182$ .

Under identical imaging conditions ( $T, k, N_o$  and  $t$  equal), the ratio of sensitivities needed by the two devices will be, using Equation 8,

$$\frac{S_{sc}}{S_{Ge}} = \frac{(C^2 - k^2\beta^2)_{Ge}(1 + C)_{sc}}{(C^2 - k^2\beta^2)_{sc}(1 + C)_{Ge}} = 5.4$$

while the actual ratio is just 1.9. Thus, while the scintillation camera is almost twice as sensitive as the HPGe camera, under the given imaging conditions it needs 5.4 times more counts than the HPGe camera to achieve equivalent signal to noise.

It can also be appreciated from Equation 9 that as a planar imaging device the scintillation camera is operating with an imaging efficiency of 14 percent, while the HPGe camera reaches an imaging efficiency of 74 percent. Neither device extracts all the available information, that is, neither is statistics limited. We also ask whether either instrument can achieve the signal to noise needed to resolve the lesion. For the given example the effective peak activity over the lesion is  $1.32\mu\text{Ci}$ , while it is  $0.88\mu\text{Ci}/\text{cm}^2$  over the surrounding area. If we take an area of interest of 3cm in diameter ( $7\text{cm}^2$ ), it will represent an effective activity of approximately  $6\mu\text{Ci}$ , thus  $N_o = 2.22 \times 10^5/\text{sec}$ . For  $k=2$ ,  $t=400 \text{ sec}$  and  $T=7\text{cm}^2/0.78\text{cm}^2=9$ , from Equation 8,  $S_n = 1.70 \times 10^{-5}$ . Since the actual sensitivity for the scintillation camera is  $4.8 \text{ cts/sec}/\mu\text{Ci}$  or  $1.3 \times 10^{-4}$ , this instrument can achieve the desired signal to noise. For the HPGe camera  $S_n = 3.2 \times 10^{-6}$ , which is considerably smaller than the achieved value of  $2.5 \text{ cts/sec}/\mu\text{Ci}$  or  $6.8 \times 10^{-5}$ , and which makes this instrument also operative under the given conditions.

b. The second example compares the performance of the scintillation camera and HPGe camera for Tl-201. In this case we will simulate a cold lesion of 1cm-diameter and 5cm depth in an immediate 1cm-thick area that is fully labeled with  $1.6\mu\text{Ci}/\text{cm}^2$  of activity. In modeling this perfusion imaging study in the heart we will neglect chest-wall background and activity, we will consider only the myocardial

walls proximal to the camera face (5cm depth) and distal to the camera (15cm depth). This far wall produces an effective output equivalent to  $0.12 \mu\text{Ci}/\text{cm}^2$ , while the proximal wall provides the main effective activity contribution of  $0.68 \mu\text{Ci}/\text{cm}^2$ . Thus, the background immediate to the lesion has an output of  $0.80 \mu\text{Ci}/\text{cm}^2$ , while the lowest activity in the lesion is  $0.12 \mu\text{Ci}/\text{cm}^2$ . This results in a peak contrast at the surface of the subject that is given by  $C_oFF=0.85$ .

In a 300 sec gated study the accumulation time would be  $t=15$  sec each at endsystole and enddiastole. For the scintillation camera this yields 450 cts in a  $7\text{cm}^2$ -area over the lesion, using a medium resolution collimator (13) and a 20 percent window (15). A HPGe camera with a medium resolution collimator will accumulate twice as many counts (13), using a narrow window encompassing the  $K\alpha_1$ - $K\alpha_2$  peaks of Tl-201. While the spatial resolution of the HPGe camera will be of the order of 5mm *FWHM* and  $SF\sim 0.1$ , the scintillation camera will be operating with a spatial resolution of 9.7mm *FWHM* (15) and  $SF\sim 1$ . *CR* for both is almost zero for the 70 keV photons. The contributions of the 135 keV and 167 keV photons of Tl-201 cause part of the increase in *SF* for the scintillation camera, but do not affect the image when using the HPGe detectors. The resulting peak contrasts, as calculated from Equation 5, will be 0.166 and 0.60 for the scintillation and HPGe cameras, respectively. As demonstrated experimentally (10), for Tl-201 scatter introduces a background that is structured, with frequency components similar to those of interest. Consequently, a conservative estimate is  $\beta\sim 0.05$  for the scintillation camera, while because of its scatter rejection and digital position readout  $\beta=0$  in the HPGe camera. From Equation 11 we can see that for a confidence factor  $k=2$  the scintillation camera can resolve a minimum area  $a=4.1\text{cm}^2$  (2.3cm-diameter) under these conditions, much larger than the  $0.8 \text{cm}^2$  of the lesion. The HPGe camera can, on the other hand, resolve a minimum area  $a=0.14\text{cm}^2$  (0.4cm-diameter). Thus, while the scintillation camera cannot provide the resolving power necessary for this particular problem, the HPGe camera capabilities exceed those needed to perform the task.

#### Nuclear imaging, different modalities (positron and planar) and radiopharmaceutical

We compare the results of planar imaging for the brain model above with what can be realized with a commercially available positron camera using an optimal positron emitter, the positron camera's parameters based on data of Reference 16. A comparison of this sort can be based on a number of criteria. For instance, the parameters  $N_o$  and  $t$  need not be the same for both cases, even though it is tempting to perform this comparison for  $t$  and  $N_o$  weighted so that the radiation dose delivered by both radiopharmaceuticals is equivalent. For cameras that image a slice at a time, the time  $t$  should be that used for examination of the full organ of interest, in this case, the brain. Even so, it is also meaningful to perform this comparison under conditions of "standard practice." In fact, "standard practice" comparisons are valid and valuable. For instance, the dose from the positron

emitter may be lower than that from the Tc-99m but larger dosages may not be possible because of instrument saturation or limits on production set by cyclotron and labeling parameters. More important, the dose-time product may have become standardized by empirical observation that image quality is not improved for larger dose-time products (8). This indicates that an instrument is no longer operating in a statistics-limited region. On the other hand, an improved device may be able to take advantage of the quanta provided by larger dosages or longer imaging times. When such is the case, comparisons should be performed for the "optimal" practical values of  $N_o$  and  $t$ .

Using the quantities specified above for the brain model, and Equation 13, we find that for the scintillation camera used in a planar imaging mode  $k=4.5$ . Similarly, for the HPGe camera  $k=9.3$ . With these values as a benchmark, we estimate  $k$  for a positron camera under conditions of "standard practice" (16). We postulate a typical study with  $10^6$  counts over a 19cm-diameter field of view, with an average count density of 2820 cts over a 1cm-diameter region. For a reconstruction where the pixel size is matched to this diameter (355 pixels) we expect (17) the product  $Mx$ -Poisson noise to be approximately 0.1. This agrees with the value measured for the positron camera (16) for objects of 19 cm-diameter and  $10^6$  counts, except that noise measurements were made for areas of the order of  $3 \times FWHM$ , or 4.5 cm-diameter. This would yield a granularity of the order of 40 percent-cm if the granularity function is well behaved (17). In estimating  $k$  for the brain model under discussion we will use noise figures of 0.1 and 0.4 to provide a range for the result. Using line sources the spatial resolution was measured to be 1.57cm *FWHM* for the sharp part of the LSF, but the shape is not Gaussian and contains an appreciable tail to the edge of the field of view, this tail due to the reconstruction process and random events. Given that a cylindrical cold lesion of 1cm-diameter appears in the output image as containing approximately 82 percent of the average counts found in the background (16),  $C/C_o=0.18$ . The effective spatial resolution can be calculated from Equation 6b, yielding  $FWHM_{eff}=1.86\text{cm}$  for 1 cm objects. Because Equation 2 is not valid below 20 percent contrast modulation, the value of  $C/C_o$  for a sphere has to be calculated from the exact expression (6). A useful approximation in this case is to treat the spherical lesion as a cylindrical one of 1cm diameter and 1 cm height, with the axis orthogonal to the image plane. Thus, the contrast modulation by effective spatial resolution will be 0.18, the factor  $C_oFF=3.60$  and the output contrast  $C=0.18C_oFF=0.65$ . Finally, while  $\beta$  was not determined explicitly, it is most likely contained in the noise figures that were measured. From Equation 13,  $k=1.1$  to 4.5: Although the positron camera eliminates the effects of over- and underlying activity, its large slice thickness and poor spatial resolution degrade contrast to the point that the modeled 1cm lesion is better detected in a planar imaging mode.

#### Interdisciplinary comparisons

We will apply Equation 13 to the brain model used as a benchmark in the prior analysis. Imaging performance will

be evaluated for a state-of-the-art x-ray CT scanner (GE 8800), and for two devices for which only data from prototypes exist: A HPGe single-photon emission computed tomography (ECT) imager (18) and a nuclear magnetic resonance (NMR) camera (5). The (unsupported) assumption is made that larger devices would maintain currently achieved specifications. The CT scanner has a spatial resolution of 1.8mm *FWHM* and a granularity of 0.5 percent-mm for a 1cm slice-width (19). The NMR camera in its present format has a spatial resolution of  $2 \times 0.5$ mm *FW* and a slice thickness of 8.4mm, with a granularity of 5 percent-mm at NMR intensity midrange.

The prototype of the HPGe ECT imager shows a spatial resolution at depth of 4mm *FWHM* in the axial plane and 2mm *FWHM* in the longitudinal plane (slice thickness). The sensitivity in this mode is 2.5 cts/sec/ $\mu$ Ci for Tc-99m. We assume that a single camera is rotated about the subject, that imaging is performed in 18 min, and that the mean absorption path is 10cm of tissue. Based on these parameters, and assuming a lesion content of 1 $\mu$ Ci of Tc-99m, 680 counts will be accumulated in the region of the lesion if 5 slices (each 2mm wide) are added. For a reconstruction performed over a 20cm-diameter cylinder with a  $2 \times 2$ -mm<sup>2</sup> pixel,  $M \sim 10$ , resulting in a noise = 0.4. As previously shown (6),  $\beta = 0$ . For a 10:1 lesion to background tissue label, 68 cts/cm<sup>3</sup> will be found in the area contiguous with the lesion, thus noise = 1.2. This is the value that is used in Equation 13.

The final question is that of  $C_o$  for each modality. If we assume that the lesion offers contrast of the type found between fat and brain tissue,  $C_o = 0.035$  for CT (19), and  $C_o = 1$  for NMR. For the nuclear study  $C_o = 9$  and for a 4mm *FWHM* resolution contrast is degraded to  $C/C_o = 0.85$  for a 1cm sphere. Questions not addressed here are motion, placement of the slice with respect to the object (we will use  $FF = 0.7$ , which results from felicitous placing), and dispersion in the parameter  $C_o$ . While the latter two are amenable to analysis, the effects of motion can only be considered in the context of the imaging problem. CT is seriously affected but is the fastest of the modalities. In NMR as developed at UCSF (5), motion will tend to produce a shift in the image. A blur is expected in the nuclear image. With these considerations in mind, and setting  $\beta = 0$ , we find that  $k(\text{CT}) = 37$  for an object of 1 cm-diameter,  $k(\text{NMR}) = 84$  and  $k(\text{ECT}) = 1.6$ , the low value due mainly to noise.

## DISCUSSION

Rose's model has been extended to deal with the principal sources of noise found in diagnostic imaging. The model provides an understanding of the effects on the output image of the physical parameters of the object/instrument system. It also permits an analysis of performance that is based on the characteristics of the output image only. Comparisons among different modalities can be made for any desirable source configuration, thus freeing the results from the fallacies that are incurred when unrealistic objects such as bar patterns are used to evaluate a device. An apparent weakness of the model is that it does not yield a single figure of

merit or performance index to characterize an instrument. Although pleasant, the belief that such an imaging number or universal curve of performance can be generated is simplistic within the context of the complexity of the diagnostic problem.

The model is limited to consideration of signal-to-noise only. It neglects factors such as the type of information provided, availability, convenience, reliability of operation of the instrument, physician and patient acceptance, cost, etc. Consequently (and fortunately) it does not pretend to have utility in selecting a certain modality for a given diagnostic study. Rather, we believe that this kind of analysis has utility in allowing us to understand the effects of the physical and engineering characteristics of the object/instrument system on the characteristics of the output image. It also has utility in helping to determine what parameters are of importance and need to be improved in future instrumentation. For instance, even the simple examples presented here, using relatively large benchmark lesions, point toward the importance of spatial resolution in diagnostic applications.

Finally, the model allows for quantitation of the potential of any one modality for obtaining information from the subject. Only when an instrument is working at its true "statistics limited" level can we say that no further improvements are possible in a particular imaging situation.

## ACKNOWLEDGMENTS

These investigations were supported in part by USPHS NIH Training Grant CA09158 from the NCI, by a grant from the Radiology Research and Education Foundation, and by NSF Grant APR 73-03161 A01. The authors are grateful for valuable input provided by Dr. F. Soussaline, Dr. R. S. Hattner and Dr. D. P. Boyd.

## REFERENCES

- Atkins, F. B. et al., "Dependence of Optimum Baseline Setting on Scatter Fraction and Detector Response Function," in *Medical Radionuclide Imaging I*, IAEA, Vienna, (1977) p. 101.
- Kaufman, L., "Nuclear Medicine Imaging," in *Medical Imaging Techniques*, Edited by K. Preston et al., Plenum Publishing Co (1979) p. 263.
- Kaufman, L. and Hattner, R. S., "Comparison of Medical Imaging Modalities: Clinical Realization and Engineering Potential of Nuclear Imaging," *SPIE* 206, 27, 1979.
- Boyd, D. P., "Status of Diagnostic X-Ray CT: 1979," *IEEE Trans. Nucl. Sci.*, NS-26: 2836, 1979.
- Crooks, L. C., et al., "Tomography of Hydrogen With NMR, and the Potential for Imaging Other Body Constituents," *SPIE* 206, 120, 1979.
- Shosa, D. and Kaufman, L., "Methodology for Evaluation of Diagnostic Imaging Instrumentation," (submitted for publication).
- Rose, A., "Vision: Human and Electronic," Plenum Press (1974) Chapter 1.
- Patton, D. D., "Controllable Parameters in Nuclear Medicine Images," *SPIE* 127:60, 1977.
- Budinger, T. F., et al., "Emission Computer Assisted Tomography With Single-Photon and Positron Annihilation Photon Emitters," *J.C.A.T.* 1:131, 1977.
- Narahara, K. A., et al., "Myocardial Imaging With Tl-201," *J. Nucl. Med.* 18:781, 1978.
- Lewellen, T. K., et al., "A Field Procedure For Quantitative Assessment of Gamma Cameras," presented at the Second Annual Western Regional

- 
- Meeting, Society of Nuclear Medicine, Las Vegas, Nevada, October 21-23, 1977.
12. Swann, S. J., et al., "Optimized Collimators for Scintillation Cameras," *J. Nucl. Med.* 17:50, 1976.
  13. Kaufman, L., et al., "Imaging Characteristics of a Small Germanium Camera," *Invest. Radiol.* 13:223, 1978.
  14. Hoffer, P. B., et al., "Measurement of Scatter Fractions in Liver and Brain Scans Performed With a Gamma Camera," *J. Nucl. Med.* (Abstract) 16:535, 1975.
  15. Hines, H. H., et al., "Spatial Resolution for Tl-201 as a function of Window Width," presented at the 4th Annual Western Regional Meeting, Society of Nuclear Medicine, Monterey, California, October 19-21, 1979.
  16. Soussaline, F., et al., "Potentials of Quantitative Methods in Positron Emission Tomography," preprint, CEA, Service Hospitalier Frederic Joliot, Orsay, France.
  17. Hanson, K. A. and Boyd, D. P., "The Characteristics of Computed Tomographic Reconstruction Noise and Their Effect on Detectability," *IEEE Trans. Nucl. Sci.* NS-25: 160, 1978.
  18. Ortendahl, D. A., et al., "High Resolution Emission Computed Tomography With a Small Germanium Camera," *IEEE Trans. Nucl. Sci.* (in press).
  19. Boyd, D. P., Private Communication, October 1979, San Francisco, California.





# Balancing processor shares of scheduling classes through controlled allocation of memory

by K. V. SASTRY

Sperry Univac  
Roseville, Minnesota

## INTRODUCTION

In a paged, virtual memory computer system used by several different classes of users simultaneously, it is reasonable to expect that each class will demand certain service rates. Service rates for a particular class can be affected by having each class use its share of (a specified percentage) Instruction Processor (IP) time at regular intervals. There are also two other important resources in the system—main memory and the IO system. The service rate of a class is very much dependent upon the availability of (or lack of) these two resources even if a processor is dedicated to a class. On the other hand, a specified IP share can be achieved by using an “unfairly large” portion of main storage for an “unfairly large” population of processes in the READY state. Although it may be favorable to do so for a particular class, it may, in fact, prevent the other classes from achieving their processor shares because of lack of enough storage.

### Problem statement

The problem then is threefold—

1. How do we ensure that a proper set of processes in each class is in the READY state so that the IP share for each class is satisfied?
2. How do we ensure that main storage is available to contain the proper set of processes mentioned in (1)? In other words, how do we map the requirement for residence of the proper set into availability of main storage? This question is especially pertinent in a class-oriented scheduling scheme.
3. What measures do we take to ensure that the direction of resource distribution is from those classes that are meeting or exceeding their IP shares through “unfairly large” allocation of resources or “unfairly large” population of processes, to those classes that are not satisfying their IP shares?

## DETERMINATION OF THE MULTIPROGRAMMING SET

The multiprogramming set is that set of processes waiting for IP or IO Service *and* having main storage allocated. In the algorithms described below, the multiprogramming set will be determined based upon the *expected* storage requirements of a process before it is rescheduled. We will here use a memory management policy based upon the concept of locality of page referencing [1]. In a paged virtual memory system the main storage requirements of a process can be represented by its working set [1,2]. It is defined to be the set of distinct pages referenced by a process in the last time interval  $T$ .

We will identify [3] three states for a process to be in. These are as follows;

**ACTIVE**—processes whose working sets are in main storage and are ready to be dispatched, except when they are waiting for page fault to be resolved or a data resource to be released. A process in this state is dispatched until its quantum runs out, or a force-deactivate occurs.

**STANDBY**—a set of processes eligible to be dispatched.

**BLOCKED**—a set of processes waiting on an external condition; e.g., a tape mount or a terminal input.

We will also assume that the system is a paging machine with virtual memory. Movement of processes into the ACTIVE set is caused and controlled by the memory balance (or imbalance) condition defined below.

Let  $C$  be the total number of scheduling classes in the system. Also, let the scheduling characteristic of class  $i$  be denoted by  $(m_i, p_i, T_i)$ , where  $m_i$  is the memory requirement of processes in class  $i$ ,  $p_i$  and  $T_i$  are the IP and IO system share which class  $i$  requires in order to satisfy the required rate of progress and response time averaged over all processes in the class. The memory balance condition is denoted by:

$$\sum_i m_i \leq M$$

where  $M$  is the total main storage size usable by all classes and  $m_i$  is the total working set requirement of all processes active in class  $i$ .

Classical working set theory [1] says that a process can be placed on the ACTIVE state if there is enough free storage available to contain its *estimated* working set.

Let  $w_{e,j}$  be the estimated working set size of a process at the time it is moved into the ACTIVE state. Let the current (actual) working set size of a process be denoted by  $w_{a,j}$ .

At the time a process is placed in the ACTIVE set, then

$$w_{a,j} \leq w_{e,j}$$

Let us assume that  $f$  number of processes are placed in the ACTIVE state in a particular class to cause processor balance and satisfy processor share of that class. Let us also assume that the memory balance condition permits us to do this, i.e.,

$$\sum_j w_{e,j} \leq (FL)$$

where  $(FL)$  is the size of the free list of page frames.

The underlying assumption here is that these  $f$  processes that are newly placed in the ACTIVE set will execute soon enough to cause:

$$w_{a,j} \cong w_{e,j}$$

During all the time that it takes these  $f$  processes to build (or rebuild) their working sets there are

$$\sum_j w_{e,j} - \sum_j w_{a,j}$$

number of free page frames that are taken. Nearly all the current virtual memory systems use this approach. Perhaps that is why the force-deactivate function is used very infrequently in those systems.

This is not an efficient approach for a class-oriented scheduling system. All the actually-free-but-unavailable page frames are unavailable for other classes to be used in order to satisfy their IP shares. To this end, a more pragmatic approach is proposed which is based on the notion of "let there be some *virtual* overcommitment of memory," that is, we activate more processes than whose estimated working sets will fit in memory. A consequential notion is "let there be some inter-class thrashing."

How do we know how many more processes to activate than the memory balance condition permits?

#### The concept of expected working set accumulation

The rate of progress of a class in real time depends upon the processor share of the class. The rate of progress of a process in a class depends upon the dynamic execution characteristics of the process when it was last placed in the STANDBY set. Nevertheless, such a process is going to page-fault until it builds its working set. This is particularly true of a newly created process. The initial storage allocation to such a process or the criteria used for memory balance to activate such a process should depend upon the fraction

of the estimated working set that the process can actually acquire before one of the following happens:

1. The process is quantum-timed-out.
2. The class is quantum-timed-out.
3. Free page frames are made available by the paging algorithm working on some process.

The probability of one of the above happening—in particular (3)—is indeed high. The reasoning is as follows.

Let  $\ell$  be the number of processes in the ACTIVE state in class  $i$  at the time a process is selected to be moved into the ACTIVE state;

- $e_j$  be the mean execution interval of process  $j$  in the class;
- $t$  be the average service time of a page fault;
- $q_i$  be the remaining quantum time of class  $i$ ;
- $q_{i,j}$  be the remaining quantum time of process  $j$  in class  $i$ ;
- $j$  be the process we are trying to select to activate;
- $p_i$  be the IP share of class  $i$  as a fraction of unity;
- $I$  be the current class;
- $Q_i$  be the total quantum for class  $i$ .

Assume for the moment that  $\ell=0$ ; then the minimum real time,  $T$ , required for the process to acquire its estimated working set  $w_{e,j}$  depends upon the following different conditions. Let  $w = w_{e,j} - w_{a,j}$

1. If  $w.t \leq q_i$ , then  $T = w.t$
2. If  $w.t > q_i$ , let  $w.t = kQ_i$  and  $n$  be the largest integer such that  $n < k$ .
  - (i) For  $k \leq 1$

The minimum real time that must elapse before  $w$  pages are acquired is increased by the dispatching time allocated to classes other than  $I$ . Depending on the monitoring granularity, this time can be computed as a function of the IP share and the class quantum of class  $I$ . For every unit of time allocated to class  $I$ , all other classes are allocated  $(1-p_i)/p_i$  units of time, so that a uniform rate of progress is achieved for all the classes. Hence

$$T = \frac{(1-p_i)}{p_i} Q_i + w.t$$

- (ii) For  $k > 1$

The elapsed real time consists again of two components. The first is the dispatching time allocated to all other classes for each full quantum of class  $I$ , which is  $(1-p_i)/p_i Q_i$ . The total elapsed time for each  $Q_i$  of class  $I$  is then

$$\frac{(1-p_i)}{p_i} Q_i + Q_i = \frac{Q_i}{p_i}$$

By definition class  $I$  receives  $n$  such  $Q_i$ 's. The second component is the remaining fraction of  $Q_i$  that must elapse to satisfy a total class virtual time of  $w.t$ . Therefore

$$T = \frac{nQ_i}{p_i} + (k-n)Q_i$$

However, the above formulas should be modified if  $\ell > 0$ . On each page fault, the process is blocked until the fault is resolved. Some other process in the class is dispatched until it faults and so on. For the purpose of analysis let us assume a simple round robin dispatching algorithm among the processes in class  $i$  that are READY for IP service. For each execution interval  $e_j$  that elapses, the class virtual time that elapses is the sum of all  $e_j$ 's. The modified time,  $T'$ , is therefore related to  $T$  as follows:

$$T' = \sum_{\text{all } i} e_i \frac{T}{e_j}$$

The contention now is that the actual storage allocation for a process being activated should be related to the ratio of the minimum time to acquire the estimated working set if there were no other classes to the real time it takes to acquire the same working set. Taking also into account the remaining quantum time for the process in question then, the actual storage allocation should be:

$$w_{a,j} = \frac{\min(w.t, q_{t,j})}{T'} w_{e,j}$$

This scheme will not only insure an equitable distribution of available storage to the processes in the class, it will also allocate only enough storage that the process can actually use.

#### AN INTER-CLASS ALGORITHM FOR CORRECTING MEMORY OVERCOMMITMENT

Memory overcommitment occurs when a process in the multiprogramming set page-faults and there are no free page frames available. In keeping up with the working set concept of memory management free page frames must be made available by deactivating one of the processes in the multiprogramming set. While there may be several processes qualified to be the deactivation candidates, we must ensure that the deactivation of a process occurs in a proper class.

It was previously pointed out that we do not want a class to achieve its IP share by merely having an "unfairly large" population of processes in the ACTIVE state. On the same token, we do not want a class to overcommit memory to itself thus preventing other classes from achieving their processor shares due to lack of storage.

To this end, let us introduce a parameter called the concurrency factor. The concurrency factor  $f_i$  for a class  $i$  is defined as the average number of processes that the site administrator or the subsystem designer deems necessary to be in the ACTIVE state such that a specified processor share and response time for that class can be achieved. For the same reason that it is possible to specify processor shares, it should be possible to specify concurrency factors.

Let  $p_i$ ,  $T_i$ ,  $f_i$  be the processor share, page traffic rate and concurrency factor respectively specified for class  $i$ .

Also let  $p_i'$ ,  $T_i'$ ,  $f_i'$ , be the current processor share, page

traffic rate and concurrency factor respectively achieved by class  $i$ .

#### Algorithm

On memory overcommitment due to a page fault in class  $I$ ,

- 1) If  $p_i' < p_i$ 
  - then (i) If  $f_i' < f_i$ 
    - then select class  $j$  such that
      - (a)  $[p_j' - p_j]$  is maximum
        - (a.1) and  $f_j' > f_j$
        - (a.2) If however  $f_j' < f_j$  for all  $j$  then select class with  $\max[p_j' - p_j]$
      - (b) If  $p_j' < p_j$  all  $j$ 
        - then (b.1) select class  $j$  with  $\max[f_j' - f_j]$
        - (b.2) If  $f_j' < f_j$ , all  $j$ 
          - (b.2.1) select class with  $\max[T_j' - T_j]$
          - (b.2.2) If, however,  $T_j' < T_j$  for all  $j$ , then select class with  $\min[f_j' - f_j]$
    - (ii) if  $f_i' > f_i$ 
      - then select class  $j$  such that
        - (a)  $[p_j' - p_j]$  is maximum
          - (a.1) and  $f_j' > f_j$
          - (a.2) If however  $f_j' < f_j$  for all  $j$  then select class with  $\max[p_j' - p_j]$
        - (b) If  $p_j' < p_j$  all  $j$ 
          - then (b.1) select class  $j$  with  $\max[f_j' - f_j]$
          - (b.2) If  $f_j' < f_j$ , all  $j$ 
            - then select class  $I$ .
    - 2) If  $p_i' > p_i$ 
      - then (i) If  $f_i' < f_i$ 
        - then select class  $j$  such that
          - (a)  $[p_j' - p_j]$  is maximum
            - (a.1) and  $f_j' > f_j$
            - (a.2) If however  $f_j' < f_j$  for all  $j$  then select class with  $\max[p_j' - p_j]$
          - (b) If  $p_j' < p_j$  all  $j \neq I$ 
            - then select class  $I$
        - (ii) If  $f_i' > f_i$ 
          - then select class  $j$  such that
            - (a)  $[p_j' - p_j]$  is maximum
              - (a.1) and  $f_j' > f_j$
              - (a.2) If  $f_j' < f_j$  all  $j \neq I$ 
                - then select class  $I$ .
            - (b) If  $P_j' < P_j$  all  $j \neq I$ 
              - then select class  $I$

Note that in the above algorithm, we are in general trusting the  $p$ 's more than the  $f$ 's. It is only when  $p' < p$  that we are choosing a class with a maximum  $f$ -deviation. While a class may maliciously specify large  $f$ , all classes together cannot specify more than 100 percent of the available IP time. Thus, the  $p$ 's are inherently more trustworthy than the  $f$ 's. Since the deviations in the IP shares are corrected through adjustment of the  $f'$  (prime)s, we have a situation that is in-

herently self correcting. Thus, while the scheduler will select the class to force—deactivate a process from, it is up to the class or subsystem scheduler to select the process to force—deactivate.

## CONCLUSIONS

In a multi-application environment consideration must be given to user processing requirements in addition to optimization of resource utilization. These processing requirements can be specified to the system in terms of percentages of IP, IO and memory resources, for each class of users. The system in turn will translate these requirements into scheduling parameters and monitor the progress made by each class in achieving its stated goal. We have developed algorithms to achieve these goals by first activating a process to satisfy a memory balance condition based on only the *expected* use of memory before this process is scheduled again. Secondly, when an imbalance occurs in the use of resources by the classes, a class and a process within the class is selected to be deactivated from the system based upon the degree and type of imbalance in resource use. Thus the al-

gorithm is inherently self correcting and provides a dynamically adjusted path to the solution. Although the algorithms are developed for a paged, virtual memory system, they can be applied to non-paged systems as well with somewhat longer degree of imbalance in resource use. The equations to compute the expected use of main storage can be made more accurate by using distributions of quantum times for classes and more complex IP service disciplines.

## REFERENCES

1. Denning, P. J., "The Working Set Model for Program Behavior," CACM, Vol. 2, No. 5, May 1968, pp. 323-333.
2. Denning, P. J. and Schwartz, S. C., "Properties of the Working Set Model," CACM, March 1972, pp. 191-198.
3. Sastry, K. V., "Process Management in a Paging Machine," COMPSAC '79, November 5-8, 1979, Proceedings pp. 198-204.
4. Rodriguez, J., Rossell, "The Design, Implementation, and Evaluation of a Working Set Dispatcher," CACM, April 1973, pp. 247-253.
5. Ellison, C. M., "The Utah Tenex Scheduler," Proceedings of the IEEE, Vol. 63, No. 6, June 1975, pp. 940-945.
6. Bernstein, A. J. and Sharp, J. C., "A Policy-Driven Scheduler for a Time-Sharing System," Communications of the ACM, Vol. 14, No. 2, Feb. 1971, pp. 74-78.
7. Private Correspondence with G. E. Newton and J. R. Jordan.

## Management in Data Processing

This group of sessions is directed toward management in data processing and consists of three presentation discussion sessions and two panel sessions. The sessions have been selected for their appropriateness to today's problems in data processing management.

The first part of "Change Management" deals with the problems associated with the preparation and planning required to implement a data processing system (new or renew) and how to cope with the change. The second part will discuss the advances women have made in data processing management and their roles in change management.

Potpourri, for want of a better title, will be a panel discussion concerning computing careers and education: deciding exactly what you want to do in life and where you want to do it, and related topics.

"Computer Aided Systems Integration" will be a panel discussing the reality, dilemma, or myth of this topic. The panel will include such notables as Samuel C. Phillips of TRW, Dan Roman from George Washington University, and Albert Rubenstein from Northwestern University. This panel hopes to be catalytic and focus industry-wide attention to the task of identifying, processing, and displaying "System Integration."

"Management" will focus on two presentations. The first will be concerned with new applications for data entry and will discuss the various aspects and techniques of on-line data entry and how these effect management. This portion will range far afield from the collection of inventory data in a supermarket to the latest techniques in voice input. The second portion of this session will be directed toward the future management concerns regarding office automation. Attention will be further focused on methods to make top management aware of the new technologies in word processing and how to sell the idea.

"Management Performance" is a session in conjunction with the half-day workshop and is a must for the workshop participants. It will deal with why managers succeed, among other topics. Attendance should not be limited to workshop participants, however.



John C. Biddle  
*Area Director*



# Applications of exemplary programming

by WILLIAM S. FAUGHT

*The Rand Corporation*  
Santa Monica, California

## INTRODUCTION

Is there an easy way for a computer user to create new software?

One of the main obstacles to effective computer usage is the difficulty of developing software to perform a new task. Computer users, once they discover or identify a task, are faced with the difficult job of software specification and development. The current sequence of task analysis, program requirement specification, coding, and debugging has four bottlenecks: (1) if the user is not a programmer, he must find one; (2) the programmer must translate his mental model of the task into an algorithm; (3) the programmer must translate the algorithm into appropriate programming language statements; (4) the programmer must specify and debug the details of the algorithm, such as initial specifications, branch conditions, and terminating conditions.

In this paper, we discuss programming by example, and show how it can be useful in solving the above problems in three applications: (1) as a software generator for non-programmers in a specific domain; (2) as a specification technique for programmers (DWIT); and (3) as a specific software tool for program development (AUDIT).

We first give a brief overview of the EP paradigm and the EP system. We then describe three application areas in which programming by example could help solve software specification problems. Next, we characterize the class of tasks for which programming by example is suitable. Finally, we analyze how programming by example helps solve software specification problems in general.

### *The EP paradigm*

In the EP project, we have turned the normal programming sequence around. Instead of specifying an algorithm, a program, and an example for debugging, the programmer specifies an example, and the EP system infers the algorithm and the program; i.e., the programmer specifies the program only by giving an example of the desired sequence of actions the program is to perform. The idea is that the user shows EP how to do a task by performing the task himself.

The paradigm is as follows: The user performs some task on a computer, e.g., transferring a file from computer to computer, retrieving information from a data base, or au-

diting a data file. EP watches over the user's shoulder, creating a trace of the interaction between the user and the system he is using. When the task is done, EP constructs an algorithm or model of the interaction. Part of this construction may involve questions to the user or advice from the user. EP then constructs a program from the model and stores it in a library for subsequent use.

As EP watches, it builds a trace or protocol of the interaction. The trace is a verbatim description of the interaction. EP stores the trace to use with an editing facility for correcting mistakes in the protocol. All advice from the user also enters the trace.

From the trace, EP constructs a model of the interaction. The model can be thought of in two ways: as a generalization of the trace or as an interpretation (or explanation) of the user's actions. For example, if the trace contains a sequence of repetitive actions, the corresponding structure in the model might be a FOR or WHILE loop. User advice both directs the model construction and constrains the space of potential explanations. EP can either generate an agent (a program) in a particular language to perform the task, or interpret the model directly to perform the task. EP stores the trace, model, and agent in a library for later use.

We plan to extend the current EP system so that models (and agents) can be modified with multiple examples of protocols. In that case, the user provides additional protocols in two ways: (1) the user performs an entirely new protocol, creating a new trace and model, and then tells EP to consolidate the two examples into a single model; (2) the user performs the old model in a "careful" mode, verifying each action before EP performs it; when a change is required, the user takes over from EP in performing the task; EP then consolidates this new (partial) protocol with the old trace and model.

EP can also aid the user in performing repetitive tasks. If EP is watching the user perform a sequence of steps in a repetitive task, the user need only perform enough steps until EP can construct a model for the repetition (perhaps lacking an exit condition). The user can then ask EP to perform the remainder of the sequence, optionally verifying each step as it is performed.

The EP paradigm has two implicit assumptions. The first is the requirement for a performance language. The user must demonstrate the task to EP by performing the task in-



teractively in real time on a computer. Most operating system tasks on interactive computers have a language and facility for doing this. Interpreted languages such as interactive INTERLISP also provide this capability.

The second assumption involves the user's expertise. EP will be designed and used differently depending upon the user's programming expertise. There is an implicit assumption that the user will be able to verify the resulting model or agent if EP presents it to him in a suitable form. This will be true for an expert user, and the expert can give EP sophisticated advice to write complex programs. However, the non-expert user may only be able to specify the task by performing it. An algorithmic or other formal representation may be foreign to him. EP may have to present the novice user's actions to him at a high level because it cannot rely on user advice or verification.

## USER INTERACTIVE TASKS

User-interactive tasks are those in which a person uses a computer operating system and its applications programs directly to accomplish some tasks. Several examples are:

1. computer network tasks
  - a. transferring files
  - b. logging in to remote systems to read mail
2. operating systems tasks
  - a. file maintenance
3. data base retrievals
  - a. periodic retrieval of time-oriented information
4. edit macros in one- and two-dimensional editors
5. tutorials.

There is a need for writing programs to free computer users from these tasks. The tasks tend to be repetitive and detail-prone. Computer users tend to be inefficient at such tasks. Further, these tasks are labor demanding: The computer user must (typically) wait for the computer to complete each step before typing in the next action. Delays can take several seconds up to a minute. Meanwhile, the user's attention is held in limbo: No other tasks can be started because of the unpredictability of the computer's response, but the response can be delayed enough to lose the user's attention and increase the likelihood of his errors.

The characteristics of these tasks make it difficult to justify writing a separate program for each task. The tasks tend to be (relatively) fast-changing. A manager may want information on accounts receivable from Utah for one month and Nevada the next. Or a systems maintainer may have different features he wishes to exercise each day, but the features change each month. The tasks also tend to be person-specific. Each person has his particular data base item to retrieve, or remote site to retrieve mail from, or edit macros to write. Finally, most of the tasks involve complicated, detailed input/output specifications to interact with the various computer systems. It would be difficult to justify writing (and rewriting) programs for such tasks, and most programmers would dislike writing such programs.

EP, however, seems ideally suited for creating software for these tasks. With EP, users can create their own software without a separate specification loop. The EP paradigm assumes that the user will be task-intelligent and program-naive: The user knows how to do the task, but does not know how to manipulate programming constructs and code to build a program. With EP, the user simply shows an example of his task; EP builds the program. He then has a set of personalized agents for accomplishing his tasks. Further, tasks with simple control structures are easy to demonstrate. The programs needed to accomplish the tasks tend to be relatively simple, consisting of straight-line code, branches, loops, and variable instantiation. There are few complex control structures necessary like parsers or production systems. (Note that the user would find it difficult to demonstrate such control structures to EP.)

An additional bonus to the EP paradigm in this application is the notion of capturing expertise. The user may not know how to accomplish a task, so he may ask an expert for advice. The expert or the expert and the user together can then perform the task with EP watching and create an agent to do the task. Later, the user has the expertise "captured" in the agent—the agent can perform the task itself. Additionally, the expert may be the user himself. The user may have had to consult a manual to perform some seldom-performed task. By constructing an agent, perhaps with comments to himself included, the user can capture his temporary expertise in the agent and free himself from remembering the details.

The current EP system has several features to aid users in these applications. The first is a library of the user's agents (programs) for storing and retrieving agents that the user created. The second is a "text" feature. The user can insert text comments into the agent for documentation. These comments are typed to the user as the agent is running, describing the agent's actions. Finally, EP has a simple knowledge base of information about the various systems that EP's users interact with. This knowledge base is a source of heuristics for inferring branch conditions and loop boundaries.

Problems we anticipate in providing users with an EP software capability center on software maintenance. Users may create software with little documentation and distribute the software to non-expert users. Also, application systems may change and require the agent to be updated. EP has several features which help alleviate the problem. First, the user can watch EP interact with the applications programs or system and can see where EP's agent is in error. Second is the text feature described above for putting documentation in the agent. Finally, the agent can be executed in a "careful" mode which asks the user for confirmation of EP messages to the system before it sends them to the system. The user can stop any invalid action before it occurs.

## *DWIT (Do What I want)*

Since the advent of programming languages, computer programmers have searched for ways to rid themselves of the compile-load-execute sequence which kept them distant from their programs. Interpreted languages and interactive

debuggers redirected the programmer's efforts from dealing with an entire program to developing a single function. INTERLISP, an interactive version of LISP, is an example of an intensive effort to provide the programmer with a "friendly" environment in which to implement his ideas in programs. Its facilities include an interactive editor, the BREAK package for interactive debugging, the DWIM (Do What I Mean) facility for correcting the user's errors in using the system, and a programmer's assistant for UNDOing and REDOing the programmer's actions by retaining a history of them (Teitelman, 1969). The main purpose of these facilities is to get the programmer "closer" to the program—closer in terms of developing the program interactively, rather than in batch mode.

However, none of these systems allows the user to develop the program completely interactively. There is still a vestige of the compile-load-execute sequence at some level. For example, in INTERLISP, the programmer must still specify an entire function before any (semantic) debugging can take place. The user is consigned to a sequence of developing an algorithm, writing the function with the editor, constructing a test environment, and running the function to debug it.

Programming by example provides a way to reduce this distance by allowing the programmer the ultimate interaction—at the program statement level. The programmer types to an interactive system with an interpreted programming language (e.g., INTERLISP). The user types each program statement to the program interpreter with appropriate arguments. The program specification system watches what the user types and creates an agent to perform the same function. When the user is done with the task, the agent is translated into the programming language and stored in the user's core image.

The important feature of the paradigm is that the user debugs each program statement as he types it in. The user verifies that the statement did what he intended as the interpreter executes the statement. When the user is done with the function, he is assured that it is bug-free, at least for the example he used (which is likely to be generic to the task). (Note that the user must be a competent programmer in the performance language).

To program a function, the user simply types the program statements to the interpreter with example arguments. By doing so, he shows DWIT both the program's data structures and the control flow. DWIT stores the data structures in a variable list. The user's task is to transform the initial variable list, consisting of the input parameters and (implicitly) all global variables, into results to be returned as the value of the function or to set global variables to. Control structure is demonstrated by the sequence of actions that the user takes. At the simplest level, the paradigm can be thought of as an extended macro feature with registers (the variable list) and with the ability to infer branching and looping control structures.

Although most of the function's actions can be shown by example, DWIT has a few commands to ease both DWIT's and the user's job. The user can demonstrate data structures precisely by his input statements. Control structures, how-

ever, require a few aids due to their implicit nature. The problem is that the user can do some actions in his mind that do not appear in the protocol. Conditions are the prime example. If DWIT were clever enough, it could infer the conditions and ask the user for verification. For simple cases, this would suffice. For complex cases, however, the user needs a way to tell DWIT the proper condition. The user does this with a special command, telling DWIT the branch point and the condition.

The second type of command helps the user with repetitive examples. The purpose is to require the user to type in a particular control path only once. When the path is to be repeated, the user can specify where to start in the path, and DWIT will continue execution from there. Options such as a "careful" mode and stopping after each loop constrain the execution.

In addition to providing instant debugging of his program statements, this paradigm aids the user in several ways. First, the user has an example at hand to prompt his actions. By seeing his actions performed in the example, he is prompted to perform the next action. Second, the user has a trace of the example's execution path that provides a history of what action has taken place. Finally, many times the user can type in just part of the action he wants to take place and DWIT can complete it.

DWIT can be given additional programming heuristics to aid the user. First, DWIT could infer some conditions at branch points, based on the syntax of program statements. Second, DWIT could know the parameter types of each function and could attempt to correct the user's statements, e.g., correcting the order of function parameters based on the parameters' types. Third, DWIT could generate some functions on its own. The user could type "LOCATE <expr>" where <expr> is a performance language expression that the current variable contains. DWIT would then locate the occurrence of <expr> and generate the function which the user could have typed to extract <expr> from the current variable.

The user's protocol provides two byproducts: documentation and a test case. The examples at each step can be inserted into the generated function as comments. Also, the initial arguments and the final returned result(s) can provide global documentation as to the calling sequence and returned value of the function. The protocol also serves as a test case for future modifications. These modifications can be tested on the example for one test of the function's correctness.

A prototype version of a feature similar to DWIT (called the Mind Channel) has been implemented in the current version of EP. Two questions remain: What types of functions are most suitable to development with DWIT, and how can DWIT provide enough documentation and flexibility so that large-scale systems can be developed incrementally. We will return to these questions later.

#### AUDIT - A PROGRAM GENERATOR FOR DATA FILE AUDITING

Two features of programming by example are useful for a program generator. One feature is instant verification of

program actions. Basically, the user attempts to do an action, sees its effects, and instantly verifies its correctness. A second feature is the specification system's attempt to do part of the specifying. The user may type an input and an output and ask the system to synthesize the intervening function.

We will argue that this type of cooperation provides a useful shortcut to program specification in certain domains. Two types of shortcuts are possible. The first concerns data structures. The specification system can deduce data formats from examples supplied by the user. The second shortcut is in control structure. If the user executes a partial program that he has developed, he need not fill in little-used branches until (and unless) he actually needs them.

The domain we will consider is data auditing. In this domain, the user has one or more data files that he wants to verify for accuracy or peruse for special cases. The file may originate from an errorful source and require format checking. Or the user may wish to compute a frequency distribution of the values appearing in various fields or verify that the values are within acceptable ranges. Statisticians typically perform several such audits on their data files before performing statistical analyses.

The current method for data auditing is to write a separate program for each type of audit, using either a general purpose language like SAIL or PL/1 or a special purpose auditing language like BRIGHT (Goldberg, 1978). The user defines the structure of the file, records within the file, and fields within the records. He then specifies the fields he wants audited, their locations, the expected ranges, and whether he wants a distribution of values. He then runs the program and examines the output, which is in a tabular form.

Specification by example provides an alternate method for data auditing. (The hypothetical specification system will be called AUDIT.) The first task is to specify the format of the data. The user reads in the first few lines or records of the file (using an interpreted performance language). AUDIT attempts to parse the records and creates a parse description which defines the format. This is an interactive task, with AUDIT making hypotheses and the user confirming or denying them. The user could name the fields for easy reference.

The user then tells AUDIT what he wants done with particular fields, e.g., whether to distribute the values or verify a range. As AUDIT reads through the data file, it looks for incorrect data formats (according to its parse) and out-of-bound values. The user has the option of stopping AUDIT whenever an error condition is found, or having AUDIT record the error and continue.

AUDIT's main feature is that it does most of the detailed calculations for field specification. Initially, it counts columns or parses fields for its first set of assumptions. As it reads more examples, it parses them, looking for format errors. If the user asserts that the record is correct, AUDIT automatically extends its assumptions about the field definitions.

The other important feature is that the user can interactively develop programs for manipulating the fields. The user could create a program to select records satisfying particular

conditions from one file and write them onto another. He can also edit the file, making the same correction on each record, or reformat the file. AUDIT saves the user effort in that it continues auditing data until new data is found. The program will stop whenever it encounters an unknown condition and ask the user to construct the proper branch condition and path.

## ISSUES OF SPECIFICATION BY EXAMPLE

We can analyze the utility of specification by example in several ways: by the advantages to the user over other software specification methods, by the classes of programs amenable to demonstrating examples, and by comparing this method to other program synthesis research.

### *Software specification problems*

In general, programming by example can help solve software specification problems in a number of ways.

First, programming by example makes specification *easier* because the user can demonstrate a task rather than specify an algorithm. The user's model may not be represented in an algorithm; he may know the task only by performing it. For example, the task representation may be in kinesthetic memory (Arnold, 1960), or in "sequential action patterns" (Faught, 1978). Even if a person learned to do a task from an algorithmic specification, his representation of the task may eventually be "compiled" into a performance-oriented representation (Dennett, 1969).

Also, in demonstrating a task, the user can depend upon context and perceptual cues from a real-time example. The example provides context by putting the world in a familiar and complete state, whereby the user can specify the next step. Further, it may be easier to specify what action should be taken in one particular case than to generalize over several cases. One attempt to accommodate this difficulty has been the development of production system languages (e.g., RITA (Anderson & Gillogly, 1976)) that enable their users to specify actions in IF-THEN rules.

Second, programming by example makes specification *more accurate* than coding in a programming language. The trace provides an example that the model must a priori account for. If this trace is correct (which may be easier for the user to verify than an algorithm) then the model must be correct for at least one example. Also, the user gets immediate verification of the correctness of his input to the system. If it is wrong, he can correct it on the spot and eliminate this source of bugs. Finally, the specification system will decide on implementation details (to some extent). If the user can perform the task on the system, the synthesized program can, at the minimum, perform the task in the same way. The program also has the option of performing some of the actions internally, rather than as side effects. The user, however, must decide how to perform the task given the real-time interactive languages available to him on the system.

Third, programming by example makes program specification *less tedious*. The motivation behind the paradigm is to acquire a program specification with the simplest and minimum input from the user. To that end, the specification system can use its accumulated knowledge to infer the specification from a minimal set of traces in two ways:

1. The system can use its knowledge of programming constructs to flesh out partial descriptions. For example, when the system detects a repetitive sequence of actions, it will hypothesize a loop. It constructs the body of the loop, as would a programmer building an algorithm. It then attempts to fill in the initial and exit conditions, using its knowledge base of programming constructs and the trace. The point is that the specification system does the hand simulation to generate the conditions, not the user.

2. The specification system can use its knowledge of the domain to supply tests for error conditions and error correction procedures, conditions that will probably never be demonstrated.

Finally, programming by example eliminates one aspect of *debugging* and provides one form of *documentation* for the program. Since the model must account for the trace, the user is assured that in at least one case the resulting program is correct. Debugging is moved to the level of program modification. The user's task is to extend the model to account for more tasks or more cases. However, he still retains the original trace to test, so that later changes must be compatible with the earlier example. These saved traces also provide a form of documentation in terms of an example created by the user and hopefully suggestive of the purpose of the program.

#### *Classes of programs amenable to programming by example*

The basis of programming by example is the completeness of the user's task example. The user must be able to demonstrate the task in a step-by-step sequence of actions on the target system. Thus, tasks which require complex control structures, such as context-free parsers and language translators, would be impossible to demonstrate. The control structure should also be natural or easy for the user to show. This probably eliminates target programs that have a control structure like a command interpreter or a production system. However, with an extension to allow the user to suggest control structures such as command interpreters, the specification system could direct the user's demonstration and learn the control structure under its own guidance. Thus, the set of programs most amenable to programming by example should be sequential, for the most part, with simple looping and branching.

The paradigm is also dependent upon the user having some performance language available to perform the task. Thus, tasks that require special purpose languages that have no interpreter, such as programming in assembler language, are not suitable.

As discussed above, user-interactive tasks seem ideally

suited to software specification by example. Users need programs to free themselves of repetitive, detailed interactions with applications programs. Yet writing such programs cannot be justified due to their fast-changing specifications. Because the user is most in tune with his needs and is closest to the problem definition, an EP-like facility can provide the user with a library of individualized software. Examples of such tasks are: computer network accessing, operating systems interactions, file auditing and maintenance, data base retrievals, editing macros, and tutorials.

One limitation of EP's application may be the limits of the performance language. In this case, EP may have to provide the user with its own performance language, such as DWIT and AUDIT.

Tasks that are rich in input/output interaction allow the specification system to do much of the user's work in formatting the data such as in AUDIT. Also, these tasks are easy to show because most of the algorithm is side effects, and therefore can be demonstrated in the performance language, such as pure functions in DWIT.

Certain task characteristics inhibit software specification by example. For instance, some tasks requiring special I/O, such as two-dimensional text editors and other graphics, are especially difficult for the specification system to understand because the user's actions may depend upon the context of the screen. The general solution to this would require the system to simulate the screen internally; the screen then becomes the context for the user's actions. The same problem occurs with standard screen or terminal conventions. For example, backspaces are handled differently by most systems. In order for a specification system to understand their effects, it must simulate the effect of the backspace on each system. Of course, this applies to all of the other editing commands on each system.

Report generation and business data processing in general would necessitate additional features to a specification system because of the complexity of the task specification. To keep from having to show an impossibly long trace, the task must be broken into functions. The system would then need additional bookkeeping facilities to help the user merge the program pieces.

For some tasks, such as matrix multiplication and sorting, it is easier to describe the algorithm than to demonstrate, step-by-step, the execution path of the required action sequence. For these tasks, demonstrating an example would be inappropriate.

#### *Other program synthesis research*

Programming by example computations has been extensively studied (cf. Bauer, 1975; Bierman et al., 1975; and Bierman & Krishnaswamy, 1976). Bierman's system, the autoprogrammer, is an excellent example of the scratchpad concept. The programmer tries out a line of (assembly language) program by typing it to the autoprogrammer interpreter and interactively debugs the line. The autoprogrammer then constructs a complete listing of the code. EP is also

related to the QBE system developed by IBM for data base retrieval. This system retrieves information based on examples of the type of information desired (Girdonsky and Neudecker, 1976).

We have extended the concept of programming by example in several ways. First, we have outlined the specific assumptions about the relative advantages of examples to other program specification media. Second, we have shown ways that the specification task can be shortened by using a knowledge base of domain and programming knowledge to fill in implicit branches and loops. Third, we have defined the characteristics of tasks that make the EP paradigm amenable. Finally, we have defined several prototype systems to serve particular known needs and analyzed how those needs can be met.

The main difference between EP and other automatic programming projects is how the user specifies a task. In EP, the user shows an example by performing the task live on a computer (as opposed to simulating what the system "would have said"). In Balzer's SAFE project (Balzer, 1978) and Heidorn's automatic business programming work (Heidorn, 1976), task specification is in natural language. In Manna and Waldinger's project (Manna & Waldinger, 1978), task specification is a series of input/output specifications that the user provides—a type of simulation. Finally, in Green's PSI project (Green, 1977), the main emphasis is on incremental program specification by multiple means, including natural language and example. As with Waldinger, the example is not live; the user must provide both input and output specification.

Another difference is in task domain. EP attempts to mesh with current computer technology to provide a useful tool for computer users. Therefore we have focused on typical interactive tasks, such as operating system, network, and file maintenance interactions.

Finally, EP attempts to provide an immediate software tool, but one which can be incrementally improved as more AI techniques are developed. Therefore, we have attempted to find techniques in which EP can use as much information as it has but require as little information as possible to be viable. This characteristic will make EP readily available for new domains.

## ACKNOWLEDGMENTS

This work has benefited substantially from comments and suggestions by Don Waterman, Phil Klahr, and Steve Tepper. The research reported here has been supported, in part, by the Defense Advanced Research Projects Agency under contract MDA903-78-C-0029.

## REFERENCES

- Arnold, M. B., *Emotion and personality*, New York; Columbia University Press, 1960.
- Anderson, R. H. and Gillogly, J. J., "Rand Intelligent Terminal Agent (RITA): Design Philosophy," Rand Report R-1809-ARPA, The Rand Corporation, Santa Monica, California, February 1976.
- Balzer, R., Goldman, N., and Wile, D., "Informality in program specifications," *AFIPS-NCC Conference Proceedings*, 1978.
- Bauer, Michael A., "A basis for the acquisition of procedures from protocols," DAI Research Report No. 10, Department of Artificial Intelligence, University of Edinburgh, 1975.
- Biermann, Alan W., Baum, R. I., and Petry, F. E., "Speeding up the synthesis of programs from traces," *IEEE Transactions on Computers*, Vol. C-24, No. 2, 1975.
- Biermann, Alan W. and Krishnaswamy, R., "Constructing programs from example computations," *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 3, 1976.
- Dennett, D. C., *Content and consciousness*, New York; The Humanities Press, 1969.
- Faught, William S., *Motivation and intensionality in a computer simulation model of paranoia*, Basel: Birkhaeuser Verlag, 1978.
- Girdonsky, M. and Neudecker, R., "Making the Computer Easier to Use," *IBM Research Highlights*, November 1976.
- Goldberg, R. N., "BRIGHT User's Manual," Department of Computer Science, Rutgers University, May 1978.
- Green, C., "A summary of the PSI program synthesis system," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, 1977.
- Heidorn, G. E., "Automatic programming through natural language dialogue: A survey," *Journal of Research and Development*, Vol. 20, No. 4, 1976.
- Manna, Zohar and Waldinger, Richard, "The DEDuctive ALgorithm Ur-Synthesizer," *AFIPS-NCC Conference Proceedings*, 1978.
- Teitelman, W., "Toward a programming laboratory," International Joint Conference on Artificial Intelligence, May 1969.
- Waterman, Donald A., "Rule-directed interactive transaction agents: An approach to knowledge acquisition," Report R-2171-ARPA, Rand Corporation, Santa Monica, California, 1978.

# Microprocessor software engineering training: a case study

by CHRISTINE L. BRAUN

*SofTech, Inc.*  
Waltham, Massachusetts

## INTRODUCTION

The rapid growth of the computer software field has resulted in a problem that is beginning to threaten that growth—a critical scarcity of skilled software engineers, trained in modern software engineering practices and able to function effectively in a real working environment. In response to this need, industrial training, through public courses, in-house programs, and audio-visual packages, has become one of the fastest-growing areas of the computer field. However, such programs frequently fail to provide the depth of training that is really required to produce skilled software engineers, and are often too general-purpose to address the particular needs of the company procuring the program.

ITT Defense Communications Division (ITTDCD) recognized this shortage of skilled professionals and realized that a serious commitment to training would be required. They determined that their needs could best be met through an in-house course, tailored to the ITTDCD working environment and designed to train existing scientists and engineers to the point where they could function effectively in that environment.

ITTDCD, together with SofTech, Inc., conceived a course to accomplish this objective, to be designed and presented by SofTech. The course was planned to run for nine weeks, covering all phases of a software project as it is conducted at ITTDCD, from analysis, through design, coding, and testing, and was oriented toward microprocessor software development, in keeping with ITTDCD's application requirements. Modern software engineering practices were to be incorporated and emphasized throughout. Because ITTDCD produces software in a DoD environment, military standards for development and documentation were to be introduced and employed. A major aspect of the course was to be a single running class project, to be followed through the entire software life cycle as it was presented in the course, thus simulating the actual working environment that the students would experience after the course.

This paper describes the curriculum developed for the course and presents our experiences in conducting it for the first time. The discussion emphasizes the attempts to parallel the actual working experience, and points out the many counterparts to "real life" that in fact occurred during the program.

## OVERVIEW

### *The students*

The course was initially presented to a group of ten scientists and engineers, all of whom had chosen to take the course and indicated a desire to transfer to a software engineering position afterward. All students had some familiarity with computer architecture, and all had previously been exposed to programming, typically at the level of one college FORTRAN course. Thus the course was geared toward students with this background.

The engineering background of the class permitted a relatively sophisticated presentation of certain topics, such as the hardware-software interactions and tradeoffs during system analysis, the microprocessor architecture, etc. It also led to selection of a class project more readily understood by engineers. This mix of hardware engineering background with training in modern software engineering is expected to produce individuals uniquely qualified for microprocessor software engineering applications.

### *Curriculum*

The nine week program is organized into four high level subject areas called "modules." These are:

- Functional Analysis
- Top Level Design
- Detailed Design
- Microprocessor Coding and Testing

Each module is further subdivided into one or more "units" of one week or less, each addressing a particular course topic. Figure 1 presents the curriculum outline, and Figure 2, is a timeline showing the phasing of the various units.

The Functional Analysis and Top-Level Design modules present only an introduction to these topics, reflecting the goal of the course to produce software developers rather than analysts. However, it is essential that a programmer understand the relationship of his activity to that of the systems analysts and designers, who define the requirements he must meet and the interfaces he must observe. The intent

Module 1 - Functional Analysis
Unit 1.1 - Introduction to Analysis
Module 2 - Top Level Design
Unit 2.1 - Introduction to Design
Module 3 - Detailed Design
Unit 3.1 - PASCAL as a PDL
Unit 3.2 - Structured Programming
Unit 3.3 - Documentation, Testing, and Debugging
Unit 3.4 - Design Reviews and Structured Walkthroughs
Unit 3.5 - Host Computer Facilities
Unit 3.6 - Detailed Design Lab
Module 4 - Microprocessor Coding and Testing
Unit 4.1 - Microprocessor Architecture and Instruction Set
Unit 4.2 - Advanced Coding Techniques
Unit 4.3 - Microprocessor Development Facilities
Unit 4.4 - Microprocessor Lab Exercise

Figure 1—Curriculum outline.

of the first week of the course is to establish an understanding of the purpose and importance of these activities, and to explain their relationship to the detailed design and development activities.

The Detailed Design module forms the heart of the training program, introducing modern software engineering practices as well as the military standards for software development and documentation. It teaches the skills and thought processes that lead to the development of modular, well-engineered software, providing significant hands-on design experience. Egoless programming and peer interaction are emphasized.

Module 4, Microprocessor Coding and Testing, teaches the actual assembly language programming of the target microprocessor. Students develop and debug programs on a host-based system providing an assembler, linkage editor, and instruction simulator for the target computer. Emphasis is placed on mapping design into assembly code in a way

that preserves program structure and modularity. Assembly level debugging techniques are also presented.

### Presentation approach

The course runs for 6 hours per day (in two 3-hour sessions) five days per week. It combines lectures, exercises, and lab sessions to permit students to gain direct experience with new concepts. Where practical exercises and labs are oriented toward the overall class project, to simulate the various aspects of a real-life project. The lecture format combines slide presentations with more informal blackboard walkthroughs, and encourages class participation and interaction at all times.

To permit evaluation and enhancement of the program for further use at ITTDCD, the first presentation of the course has been recorded on videotape. Students also completed weekly unit evaluation forms to provide an input to this process. A report evaluating the first presentation and recommending revisions and enhancements is to be prepared by SofTech, for use by ITTDCD in planning further training activity.

### MILITARY STANDARDS

A major requirement for a software engineer in ITTDCD's development environment is an understanding of military software development and documentation standards—experience that must be acquired on the job by many new employees. The training program introduces this concept from the start, explaining the requirements at each phase of the software life-cycle. The particular standard followed in the course is the Navy's MIL-STD-1679,<sup>3</sup> a new standard that will be applicable in much of ITTDCD's future business. The course briefly describes other military standards and compares and contrasts them to 1679.

MIL-STD-1679 specifies various practices to be employed in software development, and also dictates a particular sequence of documentation to be produced. During the course the students prepare (or are given) several of the standard documents for the class project system. Documents not actually developed in the course are explained in detail. Other requirements, e.g., coding standards, are practiced as the

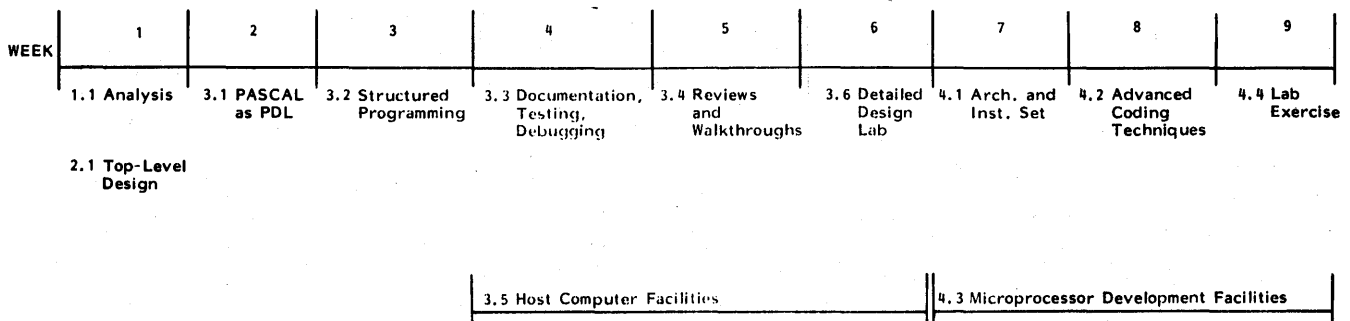


Figure 2—Course timeline.

class develops software. The military standards for project reviews, quality assurance, and configuration management are also explained and practiced. Students are also taught to read and interpret Contract Data Requirements Lists (CDRLs) and Data Item Descriptions (DIDs).

## THE DEVELOPMENT SYSTEM

The host development system introduced in the course is the PDP-11/70 under the UNIX\* operating system, a host used on several ITT projects. Running under UNIX is the Change Control Library Facility (CCLF),<sup>4</sup> a program support and configuration control tool developed by SofTech under separate contract to ITTDCD.

The target microprocessor taught is the Intel 8080. Students develop 8080 programs on the PDP-11/70 using the XAS8 Assembler<sup>5</sup> and Linkage Editor,<sup>6</sup> both developed by SofTech under separate contract to ITTDCD. Programs are debugged using the Stand Alone Emulator Package (SAEP),<sup>1</sup> an 8080 instruction simulator developed for ITT by BDM Corporation.

## THE CLASS PROJECT

The project implemented as the major exercise in the course is the Auto Tune Antenna System (ATAS), a hypothetical microprocessor-based system that involves processing representative of actual ITTDCD applications. ATAS is a system that automatically tunes a submarine antenna to a desired transmission frequency and maintains tuning, relieving the radio operator of the need to continually monitor and adjust the tuning. (A submarine antenna's tuning is affected by the action of the waves against the antenna.) Figure 3 illustrates the ATAS operator's console, and Figure 4 shows the system I/O interfaces.

The basic functions of the ATAS software are:

- 1) Initialize the system to its start state (lamp settings, tap position, etc.).
- 2) Read the frequency keyed in by the operator and check it for validity. (The keyboard is checked for possible input at every Real Time Clock interrupt. A SELECT keyin indicates the start of the frequency select mode, and an EXECUTE keyin indicates that frequency keyin is complete.)
- 3) Coarse tune the antenna to the selected frequency, by interpolation into an in-core table of frequencies and corresponding tap positions. Enable RF after coarse tuning is accomplished.
- 4) Monitor tuning by regularly (every 10 Real Time Clock interrupts) sampling the Voltage Standing Wave Ratio (VSWR) and taking corrective action if it is outside accepted limits for a certain number of measurement intervals. (Corrective action involves moving the tap to minimize the VSWR.) If acceptable tuning cannot

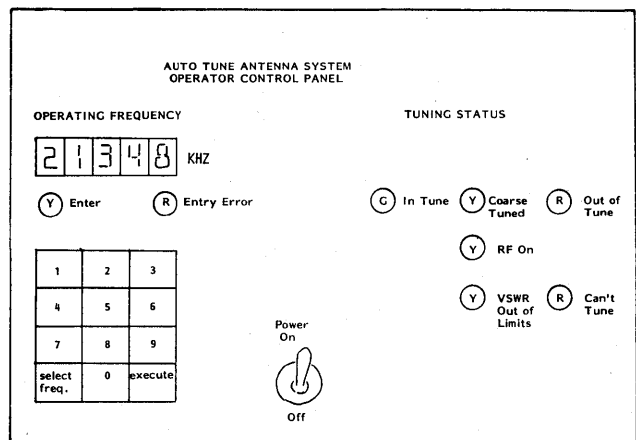


Figure 3—ATAS operator control panel

be obtained, the status lamps are set to indicate this, the RF is disabled, and tuning is discontinued. (At any time, the operator can depress SELECT and select a new frequency.)

The ATAS system introduces numerous programming concepts, including:

- interrupt handling
- foreground/background processing
- data aggregates (frequency-tap position table)
- mathematics (interpolation)
- complex flow of control (tuning adjustment)

The system is specified with a limited amount of RAM and ROM, and students are given budgets for their various subprograms. This is a realistic design constraint for microprocessor software, and encourages students to coordinate and make tradeoffs during implementation.

## THE COURSE EXPERIENCE

The following subsections narrate our week-by-week experience in presenting the course for the first time.

### *Week 1—functional analysis and top level design*

The course began with a review of computers and programming, in order to ensure a common starting point for discussion. This review included a description of various support software tools, such as operating systems, compilers, assemblers, linkage editors, and simulators.

The presentation of analysis and top-level design was then introduced by a discussion of the concept of *total system* (hardware and software) development. The role of a system engineering organization was presented, and the process of functional allocation between hardware and software was addressed. The class informally worked through the analysis of some example microprocessor-based systems (automobile

\* UNIX is a trademark of Bell Laboratories.



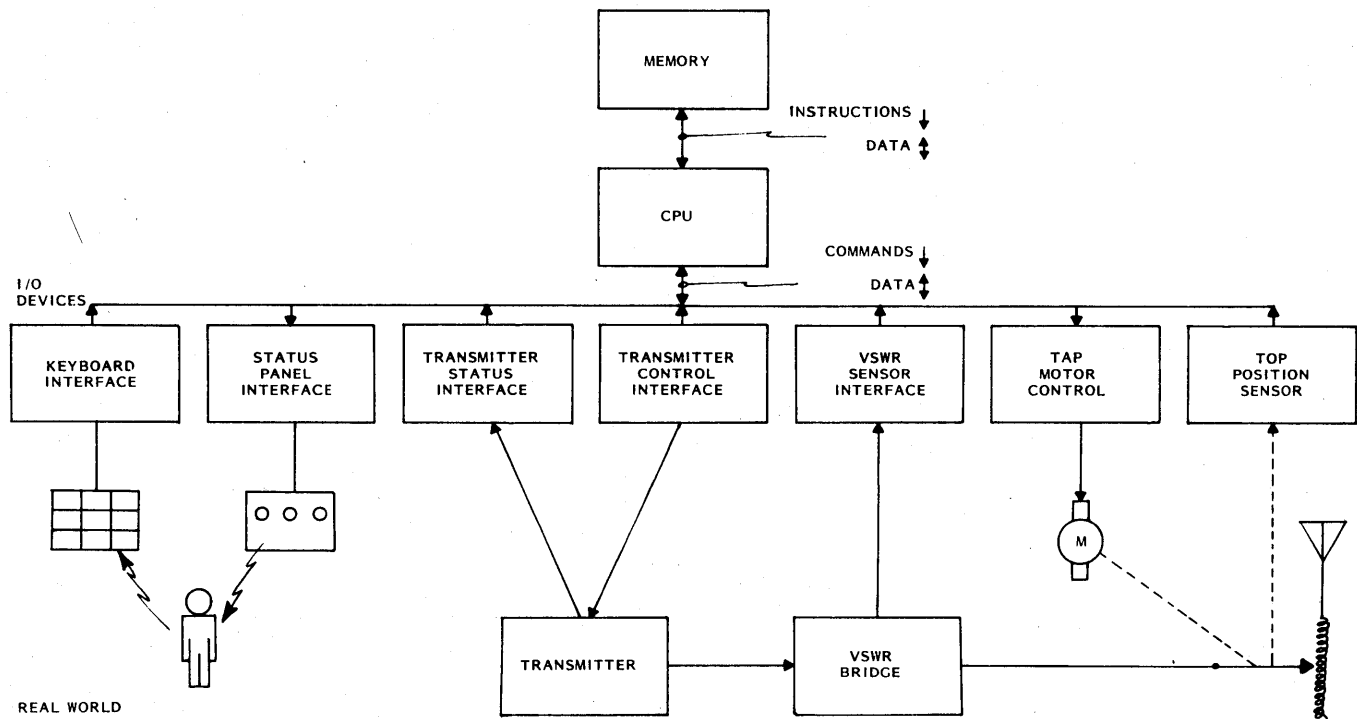


Figure 4—ATAS I/O interfaces.

cruise control, elevator controller). The ATAS system was then introduced, and aspects of the analysis of this system (performed by the instructor staff functioning as "system engineering") were presented.

Discussion then turned to the phases in the software development life cycle (analysis, top level design, detailed design, implementation, test and maintenance), and MIL-STD-1679 was introduced by describing the requirements it imposes on each phase. Students were given the ATAS Program Performance Specification (PPS) that the instructors had prepared, and were taught how to interpret and work from a PPS, and how to interact with systems engineering regarding problems or questions on the PPS. Further emphasis was then given to the role of the Military Standard, and to the other requirements it imposes on the analysis and top-level design phases.

The week also included an overview of various methodologies supporting analysis and top-level design, including SofTech's Structured Analysis and Design Technique (SADT<sup>®</sup>),\*\* PSL/PSA, Structured Systems Analysis, HIPO, Jackson Design, and Warnier/Orr Design.

Finally, the class established a document library to be used for the ATAS project, and was introduced to the concept of the author-librarian-reader review cycle.

### Week 2—Pascal as a PDL

Because the target computer for the course was to be programmed in assembly language, it was considered particu-

\*\* SADT<sup>®</sup> is a trademark of SofTech, Inc.

larly important that students be exposed to a modern high-level language containing structured programming constructs for use as an initial design mechanism. Pascal was selected for this purpose, and was taught as a Program Design Language, stressing concepts rather than perfect syntax. As students generally had little programming experience prior to this, this week was oriented toward "how to design programs," with Pascal as the language for design. Top down design was emphasized throughout the week, with several examples of problem decomposition (e.g., Eight Queens<sup>2</sup>). Programming exercises and examples were, in many instances, designed to present concepts needed in the ATAS exercise (e.g., multiplication by shifting and adding). This week allowed substantial time for students to work on exercises reinforcing the lecture material, and to discuss them afterward.

As a by-product of this unit, students learned to read and work with both BNF grammars and syntax diagrams. Also, though it had not been planned, student interest prompted some discussion of rudimentary compiler theory. The students' hardware background seemed to make them particularly aware of the apparent difficulty of this process.

### Week 3—structured programming

This unit began introducing more formal aspects of structured programming, and continued the teaching of the programming process begun in Week 2. The theory of structured programming was presented, and the basic set of control structures was discussed. One exercise was the restructuring of an intentionally unstructured flowchart that appeared

in the ATAS PPS to describe the tuning adjustment algorithm. Structuring a program into a collection of functionally simple subroutines was also emphasized.

Various descriptive techniques for developing and expressing design were presented. These included the Data Structure Diagram for describing data structures, and the Calling Hierarchy for illustrating program decomposition. The Finite State Machine concept was introduced as a design technique, and its applicability to describing the states (frequency select, fine tuning, etc.) of the ATAS system was noted.

In the examples and exercises used during the week, new programming techniques and concepts were introduced. These included various data structures (linked lists, trees); searching and sorting algorithms, etc. Students would later select among these in designing the data structure and lookup algorithm for the ATAS coarse tune function.

#### *Week 4—documentation, testing, and debugging; UNIX*

This week involved two course units—approximately half was devoted to documentation, testing, and debugging, and half to an introduction to the UNIX system. The lecture material concentrated on the requirements of MIL-STD-1679 that had not been covered in Week 1. In addition to the other required project documentation, the week included lectures on project management requirements, configuration management, and software quality assurance. Students developed both a System Operator's Manual and a Test Plan for the ATAS system. Both documents were produced by developing an outline through class interaction and then assigning a section to each student. Though this resulted in rather unusual first drafts, it is comparable to the way documents are often produced in the real world. It was also the only realistic way to have the class produce actual documentation, as individuals could not be expected to produce entire documents. The students also developed a Software Trouble Report form to be used during ATAS development.

The testing and debugging material included both a discussion of current approaches to software development testing and debugging, as well as presentation of MIL-STD-1679 requirements for formal acceptance testing. It included a detailed discussion of testing under simulation, as this was the way ATAS (like many real microprocessor-based systems) would be tested.

The UNIX classes and labs concentrated on the basics of the UNIX file system and text editor, but provided an overview of more sophisticated features such as parameterized command files. The students used the UNIX editor and run-off (nroff) to produce their ATAS System Operator's Manual and Test Plan documents. Each typed in his section, and the instructor provided the title page and ran off the final document.

#### *Week 5—reviews and walkthroughs; CCLF*

Week 5 was also shared with the Host Computer Facilities unit. The Reviews and Walkthroughs unit introduced various

concepts relating to egoless programming and peer review, including code reading and structured walkthroughs. The objectives, mechanics, and behavioral aspects of these were discussed, and exercises in each were conducted. Walkthroughs were held for both ATAS documents produced in Week 4, and each resulted in action items that were subsequently performed to produce revised documents. In fact, review of the System Operator's Manual led to a question about the correctness of the PPS in regard to the meaning of one of the status lamps. The instructor pointed out that it was necessary to go back to system engineering for a decision, and the class put in a call to the "system engineer" back at SofTech.

The lab sessions this week presented the Change Control Library Facility (CCLF), which would be used to maintain PDL and program components for the ATAS project. Students learned to create and manipulate Software Configuration Trees, the basic structure used to organize and control development within the CCLF framework.

#### *Week 6—detailed design lab*

This was a lab week with no formal instruction, devoted to development of detailed PDL design for the ATAS project. The instructor staff developed a Program Design Specification (PDS) documenting the top-level design to be followed. The PDS decomposes the system into twelve subprograms and two libraries of common subroutines (math routines and I/O routines), and fully documents their functions and interfaces.

The class was divided into two teams, each of which would develop a complete ATAS system. This was done for the following reasons:

- It seemed to be (and proved to be) about the right amount of work per person.
- It made the number of interfaces more manageable.
- Students had an opportunity for more self-management and real teamwork, as the instructor could not always be involved.

Teams elected chief programmers, who would generally be responsible for keeping the team organized (the instructor distributed "chief programmer job descriptions"). Individual assignments were determined by the teams, with input from the instructor as to the difficulty of the various subprograms.

The teams then separated to begin design. They were encouraged to produce early written design kits for review by other members of their team, and by the instructor. Designs were then refined based on comments. After this, each team held a walkthrough. The chief programmer served as the moderator, and the instructor recorded action items but otherwise generally attempted to maintain a low profile. After the walkthroughs, designs were updated again.

Throughout the week, students used the CCLF to create Configuration Trees of their PDL, eventually producing a tree with their final system PDL. The tree would then be

expanded to include parallel nodes for source, object, etc. during Module 4.

This week was particularly well-received by the students, and provided an excellent illustration of the values of thorough early design and peer interaction. With little real help from the instructor (other than nudges in the right direction) students went from faulty individual designs that they had little confidence in, to complete system designs that they were very sure of and that were in fact virtually faultless.

#### *Week 7—microprocessor architecture and instruction set*

Week 7 began the introduction of the target microprocessor, the Intel 8080. It combined a detailed presentation of the microprocessor's register and bus structure and its instruction set with an introduction to the microprocessor development facilities. As soon as the assembly language was introduced, students began using the assembler, link editor, and loader to run simple exercises.

The XAS8 assembler does not process conventional Intel 8080 assembly language—it recognizes a more expression-oriented instruction syntax and provides the control structures needed for structured programming, such as IF-THEN-ELSE and DO WHILE. (The assembler generates appropriate tests and jumps.) Thus the students were taught to develop structured assembly language programs using the same principles they had learned with Pascal.

#### *Week 8—advanced coding techniques*

A major topic in Week 8 was mapping of Pascal (PDL) constructs to 8080 assembly language. This is particularly relevant in considering the mapping of more complex data structures and techniques for accessing them. This week also introduced coding "tricks" and time and space saving techniques, along with a discussion of when tricks should be avoided because of an adverse effect on program understandability.

Another subject addressed in Week 8 was the need for coding conventions in system development. Conventions for the ATAS project (e.g., subprogram linkage, register usage) were established and documented.

During Week 8 students began to use the lab sessions to convert their ATAS PDL to assembly language, mostly concentrating on getting the code to assemble correctly rather than on executing it.

#### *Week 9—microprocessor lab exercise*

This week was devoted to completion of the ATAS project. Students completed coding, and code-read one another's subprograms. They then used the instruction simulator to perform unit and integration testing in accordance with the test plan they developed in Week 4. During this week teams were encouraged to work with one another so that a team that had developed a working version of a particular subprogram could help a team that had not. This pro-

vided an opportunity to point out the benefit of defining subprogram interfaces clearly and then observing the specification, as one person's version of a subprogram could be directly substituted for the other's.

A final output of this week was a Program Description Document for each team's design.

## CONCLUSIONS

As the course is just concluding as this paper is being written, it is not yet possible to evaluate the performance of the students as software engineers. However, our experiences in presenting the course have made us very optimistic about this. A substantial amount of material was covered and absorbed by the students. In addition, the emphasis on military standards, on real project experience, and on development tools actually used at ITTDCD, has equipped these students with skills that most new computer science graduates lack.

ITTDCD intends to follow up the course by observing and evaluating its success at producing effective software engineers. If the outcome is positive, the course will be presented to additional groups of students. As indicated previously, the course evaluation report prepared by SofTech will recommend possible modifications based on the initial experience.

The modular nature of the course makes some modifications quite straightforward. For example:

- An introductory module could be added for students with no prior programming background.
- Most of Module 3 could be used stand-alone to teach modern programming practices to students who are already programmers.
- A different course project could be substituted.
- Another military standard could be substituted.
- A different target computer could be substituted.
- A different host development facility could be substituted.

Thus the present course material forms a baseline that can easily be tailored to fit ITTDCD's changing needs in the future.

## ACKNOWLEDGEMENT

The course described in this paper was conceived and designed by G. Sampson and Gen. J. Robbins of ITTDCD, and by Dr. R. S. Eanes, Dr. C. McGowan, Dr. L. Weissman, Dr. S. Shrier, and R. Thall of SofTech.

## REFERENCES

1. BDM Corporation, *User's Manual for Stand Alone Emulator Package/Environment Simulator: Phase 0*, BDM/W-78-295-TR, 10 August 1978.

- 
2. Dijkstra, Edsger W., "Notes on Structured Programming" in *Structured Programming*, Dijkstra, E. W., Dahl, O.-J., and Hoare, C. A. R., Academic Press, New York, 1972, pp. 72-82.
  3. MIL-STD-1679 (Navy), *Military Standard—Weapon Systems Software Development*, 1 December 1978.
  4. SofTech, Inc., *ULCS Program Change Control Library Facility User's Manual*, 1038-56.1, 20 November 1978.
  5. SofTech, Inc., *ULCS Program XAS8 Macro Assembler User's Manual*, 1038-40F, 18 September 1978.
  6. SofTech, Inc., *XAS8 Link Editor User's Manual*, 1038-48F, 18 September 1978.



# Development of a microprocessor support facility for large organizations

by BRUCE E. STOCK

*Boeing Aerospace Company*  
Seattle, Washington

and

MIGUEL A. ULLOA

*Tektronix, Incorporated*  
Beaverton, Oregon

## INTRODUCTION

Microprocessor technology has grown in a decade from simple 4-bit controllers to complex 32-bit architectures which rival the performance of mainframe computers. Applications of microprocessors today range from military and aerospace programs to consumer products and toys. The rapid growth of this field has resulted in many different types of microprocessors, based on several different technologies, being offered by a number of different manufacturers. Each of these manufacturers, and several independent companies as well, also provide support tools to aid in the development and checkout of microprocessor-based products. Evaluation kits, development systems, analyzers, high level languages, and software simulators are but a few of the available aids permitting more rapid design, integration, and debugging of systems incorporating microprocessors.

The growth in the number of support tools along with the differences in cost and capability of these tools have created a new set of problems for both management and engineering. It has become difficult to select a set of tools which will provide a level of support appropriate to the complexity of the work being done, while minimizing the cost of these resources. Additionally, the training of personnel, service and maintenance of the tools, and the eventual obsolescence of the resources provided are important issues. The problem is especially acute in large organizations where hundreds of engineers may be involved in the design of a wide variety of microprocessor-based products.

In this paper we will briefly examine the types of support tools available and the suitability of these tools to certain types of development, and finally will focus on a centralized concept suited to large scale development.

## MICROPROCESSOR DEVELOPMENT SUPPORT TOOLS

There is a great variety of hardware and software tools to support the development of microprocessor-based products.

These tools range widely in their cost, complexity, and capabilities. Low-cost evaluation and learning aids are available to familiarize designers or inexperienced users with particular microprocessors.<sup>1,2</sup> Development systems ranging from a few thousand to over forty thousand dollars come in different forms and with many options to support various development phases.<sup>3,4</sup> Development systems from individual chip manufacturers provide support for one or more of that vendor's family of chips. On the other hand, "universal" systems can support several microprocessors from different manufacturers.<sup>5</sup> To support software development, operating systems, editors, assemblers, compilers, linkers, and debugging tools are provided.<sup>6,7</sup> For hardware and software integration, support for in-circuit emulation, memory mapping, and real-time analysis is available.<sup>8,9,10,11</sup> To evaluate microprocessors and do software development on mini or large computers, cross-software and simulators are provided by software houses and timesharing networks.<sup>12</sup> These and other tools are outlined in Figure 1.

Despite the variety of support tools, in general only the most popular microprocessors (or those out in the market for some time) are supported by several of these tools. Other chips, including newly introduced ones, are supported only in a limited form. For example, some microprocessors supported by cross-software or low-cost design aids are not supported by development systems or in-circuit emulation. In cases like this, in-house design of special test tools must be considered or other microprocessor choices evaluated.<sup>13</sup>

Another aspect of present support tools to be considered is their suitability and effectiveness in different types of microprocessor development. As a general rule, users of microprocessors are involved in either one-time development, periodic new design, or continuous heavy development. Simple in-house-designed test tools, low cost development systems, and cross-software are often suitable for one-time developments because of their low capital investment. When periodic designs with few engineers are involved, then a good-quality development system capable of supporting several microprocessors can be quite cost-effective.

## Evaluation/selection tools

- . Evaluation kits, design aids, teaching tools
- . Microprocessor development systems
- . Cross-software, simulators

## Software development Tools

- . Microprocessor development systems
- . Operating systems, file managers
- . Editors, assemblers, compilers, interpreters
- . Linkers, library generators
- . Simulators, debuggers, cross-software

## Hardware development tools

- . Test and measurement tools (e.g. oscilloscopes, data generators, pulse generators)
- . Logic analyzers, microprocessor analyzers

## Software/hardware integration tools

- . In-circuit emulators
- . I/O simulators
- . Memory mapping; real-time analyzers
- . PROM programmers

## Production support tools

- . Diagnostic software
- . Microprocessor development systems
- . Special-purpose automated test equipment

Figure 1-Microprocessor support tools.

tive. However, if the task requires several engineers, then a multi-user development system or timesharing cross-software support may be more appropriate than purchasing multiple stand-alone systems.

When multiple developments involve many engineers and a variety of microprocessors, the support tools as currently provided do not offer an integrated, cost-effective answer to a large organization. Neither multiple stand-alone development systems, nor multi-user development systems, nor cross-software by themselves can satisfactorily answer the major concerns of management with regard to the effective utilization and sharing of resources.

## LARGE SCALE MICROPROCESSOR DEVELOPMENT

Large organizations doing extensive microprocessor-based product development have a number of possible alternatives to provide development support. Five basic approaches to be discussed below are stand-alone development systems, stand-alone systems with timesharing, timesharing with remote emulation, multi-user stand-alone development systems, and centralized development facilities. Which approach, or combination of these approaches, will best suit a particular organization, will depend on the organization's present facilities and on its structure and monetary constraints, among other things.

### *Stand-alone development systems*

In this approach, each project or group uses as many individual development systems as is required by the complexity of the task. Each project thus has immediate access to the resources it needs, which include the development system plus the necessary peripheral equipment and test and measurement tools. There are disadvantages with this approach. First, the number of development systems and the total cost increases rapidly as the number of projects increases. Second, resources such as line printers and storage units are duplicated many times, usually without full utilization. And third, knowledge, experience, and software are not easily shared between groups or projects.

### *Stand-alone systems with timesharing*

Here the stand-alone alternative is augmented by using either an in-house timesharing system or one of the national networks to provide additional software development and checkout capability. This approach is suitable for improving software development support without duplication of resources and for including microprocessors not supported by development systems. This is a viable alternative for organizations already committed to a large number of stand-alone systems; for other organizations, the next two alternatives can be more cost-effective. A disadvantage of timesharing is of course the decline in user response time as the system becomes heavily loaded.

### *Timesharing with remote emulation*

With the availability of in-circuit emulation as a peripheral device, it has become possible to avoid the stand-alone development system completely. In this approach, a multi-user computer is connected with emulator systems such as Tektronix 8001 Microprocessor Labs.<sup>14</sup> Complete software development is done on the computer and object code is then transferred to the emulator via telephone lines or dedicated wire. Hardware/software integration is thus available with emulation, memory mapping, and real-time analysis.

This alternative provides several benefits: resource duplication is minimized while providing extensive access to the development tools; commonality of development methods, tools, and procedures is enhanced; and software libraries can be shared among engineers. One difficulty of this approach is the necessity to provide fast, error-free transfer of developed code to the emulation stations.

### *Multi-user stand-alone systems*

Within the last year, stand-alone development systems which support as many as 6 or 8 simultaneous users have become available.<sup>15</sup> Such systems provide the advantages of multiple stand-alone systems while minimizing the duplication of resources such as mass storage and line printers.

These advantages may be offset, however, by problems such as limited line printer capacity, lack of spooling capability, limited storage space, and degradation of user response time.

### *Centralized development facilities*

All the tools necessary for development of microprocessor-based products can be integrated into a physically centralized facility and made available to a large user community as a shared resource, so that microprocessor support is shared and controlled in much the same way as data processing facilities are shared and controlled through computer centers.

The central facility will normally consist of a medium-size or large multi-user computer, mass-storage devices, peripheral equipment, one-line terminals, and hardware/software integration systems. Communication lines for remote access can also be provided. The mainframe computer should include a powerful operating system and standard software utilities plus cross-assemblers, compilers, and simulators to support a variety of microprocessors from different manufacturers. For hardware development and hardware/software integration, the center should have general purpose test equipment, in-circuit emulators, I/O simulators, and PROM programming tools.

In addition to support tools, the center should have an operating staff to provide the following support services: user access control and accountability; user assistance and problem resolution; configuration management and control; system software revision and enhancement; maintenance coordination and consumable resource stocking; coordination of new user training courses and information; and integration of new software and hardware tools to support the changing technology.

The centralized approach provides all the benefits of using timesharing with remote emulation plus additional benefits not realized with any of the other alternatives. In addition to minimizing resource duplication and providing commonality of tools and methods, the center offers other advantages: knowledge, information, and software can be easily shared among all groups and projects; utilization of hardware and software tools is kept at a high level; maintenance and service of tools is the concern of the support staff alone rather than of separate departments; and training for new users is readily available.

The major disadvantage of a centralized facility is its high initial cost but this is offset by maximizing the utilization and productivity of the capital investment in resources. Another disadvantage is the possible decline in system response time when high user demand occurs.

Of the alternatives presented above, using stand-alone development systems is currently one of the most, if not the most, popular approaches followed by both small and large organizations. However, this is the alternative that can least realize all the benefits of centralization, while at the same time it requires the highest overall capital investment. Therefore, it is imperative for large organizations to weigh these alternatives carefully. Boeing Aerospace Company has done

just this, and the result of its analysis has been the implementation of the Microprocessor Development Support Center.

## MICROPROCESSOR DESIGN SUPPORT CENTER IMPLEMENTATION

### *System overview*

In early 1978 work was begun to design and implement a development facility at the Boeing plant at Kent, Washington, near Seattle. Called the Microprocessor Design Support Center (MDSC), the facility was to provide support to any company project doing microprocessor-based design. The goal of the center's designers was to provide a facility which would: accommodate a large number of users; support a wide spectrum of microprocessors; maximize the usage of resources provided; minimize training and retraining time; use off-the-shelf hardware and software to the maximum extent; and adapt to newer and more complex microprocessors with a minimum of modification.

A centralized design based on a larger minicomputer was selected as the best approach to meeting these goals. This approach was made feasible by the availability of cross-software (cross-assemblers, linking loaders, simulators, etc.) and in-circuit emulation as a peripheral device.

The center is divided into three adjacent areas: a machine room housing the central computer and its related peripherals; a software development room containing CRT terminals in a quiet environment; and an integration room containing the in-circuit emulators, I/O Simulator, and PROM Programmer. Functionally, the computer is the hub of the system, with all other devices acting as peripherals to it.

The MDSC currently provides support for the following microprocessors: 8080, 8085, 8048, 8086, 6800, 6802, 650X, F-8, 3870, 1802, Z-80, 2650, TMS and SBP9900, Z8000, and 68000. Complete hardware and software support is not available for each, but some support, either software or hardware or both, is available for all of the chips named. An in-house development is under way to provide a compiler targetable to several microprocessors. The center also provides software support for bit-slice microprocessor development via a meta assembler on the central computer. A Control Store Simulator is being acquired which will permit download of assembled microcode to the users' bit-slice prototype as well as debug and trace capabilities.

### *Computer*

The primary requirement for the central computer and its immediate peripherals is to support a timesharing environment with a large number of simultaneous users while providing a reasonable response time to each user. To satisfy this requirement, a PDP 11/70 has been selected and equipped with a megabyte of local storage, 356 megabytes of rotating storage, 48 RS-232 ports, and a DMA channel. The installation also contains the usual magnetic tape, line



printer and card reader peripherals. A rack mounted modem set with 16 channels (12 currently active) is mounted within the main computer cabinetry to give dial-in users access to the system.

### *Software*

The top level of software on the facility is the operating system. This package permits the computer operator to configure the system and allocate resources and priorities to tasks. At the user level, it provides access to all of the installed utilities of the system.

The MDSC operating system is the Interactive Application System (IAS) provided by the computer's manufacturer. This system is well suited to the MDSC environment in that it provides concurrent running of realtime, timesharing, and batch jobs. This capability permits a user at a terminal to submit a long assembly or other job to the batch stream and then continue to edit or debug other modules while it is being processed. Concurrently, another user may be exercising a real time I/O simulation task. The operating system provides access to an editor, file management utilities, word processing software, compilers, cross software, and a library of user generated code modules. In addition there are three major in-house designed programs on the system: an Emulator driver, an I/O simulator driver, and a PROM programmer driver. The Emulator driver permits interactive communications with the in-circuit emulators, transfer of developed code to and from the prototypes and several enhancements to the basic command structure of the emulator. The I/O simulation driver provides an interactive control package which permits a user to generate data bases or data streams, to output them with selectable rates and protocols to his prototype, and to collect and display the response to them by his prototype. The structure of the program also permits closing the loop to provide a limited environmental simulation; that is, the responses from the prototype may be used to modify or generate the data stream going to the prototype. The PROM programmer driver permits computer control of the entire PROM programming and verification process, thereby minimizing the opportunity for human error. In addition, utilities are provided to partition object code modules of arbitrary word width and length into ROM-sized modules, and to provide object output media in formats compatible with production-line programming equipment.

### *Emulation*

Perhaps the greatest single aid to microprocessor integration and checkout is the technique of in-circuit emulation. An in-circuit emulator provides for control of and visibility into the execution of the user's software on the actual prototype hardware. By plugging into the CPU socket, the emulator affords these advantages without any special interface wiring or hardware required in the prototype.

Ideally, an in-circuit emulator would provide a perfect replica of the operation of the emulated chip. In practice, how-

ever, in-circuit emulators tend to suffer from one or more faults, such as:

#### **Extra delay**

The use of buffers on most, if not all, of the probe pins and the inevitable delays due to extra cable length between the emulator and the prototype causes some timing differences. These are usually quite minor and only become significant for the fastest chips when employed in designs with tight timing.

#### **Non-realtime operation**

Some timing and interaction problems only manifest themselves when the prototype is running at the full design speed. Other designs may in fact not run at all in less than realtime because certain critical tasks cannot be serviced quickly enough. Some emulators have internal delays which require wait states to be added to the microprocessor's cycles. These wait states perturb critical timing tasks and mask marginal access time problems. Other emulators may permit the prototype to run at full speed, but are then unable to provide any insight into what is happening in the prototype (i.e., no trace information).

#### **Designed-in differences**

Since the emulator must perform additional tasks in the course of its operation (such as pausing and dumping register contents to a CRT) which the actual microprocessor will not have to cope with, the emulator designer must make some decisions as to what will be presented to the prototype CPU interface during these operations. Depending upon how these questions are answered, the prototype may experience a loss of refresh signals, unexpected pulses on control lines, unexpected tri-stating of lines, or other unwanted anomalies.

#### **Expropriations**

The design of the in-circuit emulator can result in its requiring part of the prototype memory or I/O address space. In other cases, a major interrupt line (such as the non-maskable interrupt) may be used by the emulator, requiring the user to work around the difficulty in his prototype.

#### **Different fan-in/fan-out**

The use of TTL buffers at the probe causes different fan-in/fan-out characteristics from the actual MOS, CMOS or bipolar processor. This can result in learning late in the integration cycle that the prototype functions perfectly with the probe in, but not with the actual microprocessor.

The MDSC has selected the Tektronix 8001 Microprocessor Development Aid<sup>14</sup> for use as its standard in-circuit em-

ulator. The 8001 is designed specifically to be used in conjunction with a host system which provides software generation capability. It is possible to configure the 8001 to emulate any of several different microprocessors by inserting appropriate emulator-board/probe units. By using a common emulator mainframe such as the 8001, it is possible for a user who has been working with an 8085 based design, for example, to move on to a Z80 project with his entire emulator command repertoire and debug strategy intact, representing a significant reduction in retraining. It is also possible to move optional equipment resources (extra memory, Real Time Trace Modules, etc.) from one user who does not need them to another who does.

With this configuration, the 8001 is integrated into a "super development system." A user at a terminal on this expanded system can model system performance using a High Level Language, perform trade studies and run benchmarks on several different microprocessors using the simulators, and prepare trade study and program documentation. He can then generate software for any of several different microprocessors, download it to an 8001, and exercise his prototype with that software. At any time he can return to the Edit/Assemble mode to correct errors, and continue with the modified code. This process is repeated until the prototype is functioning correctly. The software is then committed to ROM, the microprocessor chip is replaced in its socket, and the prototype is released for validation/verification.

### *PROM programming*

A centralized developmental laboratory such as the MDSC does a considerable amount of PROM programming in support of the integration effort. To accommodate the wide variety of devices used, a System 19 Programmer made by the DATA I/O Corporation of Issaquah, Washington, was selected. This unit accommodates a large number of programmable devices via plug-in personality modules and has a well developed RS-232 interface to a host computer. The remote control capability via this interface permits the operation of the programmer to be controlled by an interactive software package on the host computer.

The wide variety of devices which can be programmed by the System 19 exceeds the capability of the facilities provided on stand-alone microprocessor development systems. Further, a large number of down-load data formats can be accommodated, such as Binary, Intel, Tektronix-hexadecimal, ASCII, and others.

### *I/O simulation*

The MDSC provides an I/O simulation device called the Adaptive Interface Unit (AIU) which is installed at a specific integration station. The purpose of the AIU is to provide the user with a generalized hardware interface to his prototype over which user-defined stimuli may be provided and the prototype's responses can be obtained and evaluated. Used

in conjunction with in-circuit emulation, I/O simulation provides an effective means of verifying correct prototype operation prior to leaving the lab. The characteristics of the AIU/user interface are as follows:

#### **High speed parallel I/O**

A 32-bit-wide input channel and a 32-bit-wide output channel are provided with local buffering of 4K words on each channel. Transfer rates to 1 megaword per second can be accommodated with user-clocked, AIU-clocked or handshake data transfer protocols. Differential TTL drivers and receivers are provided, although special drivers or receivers can be provided on a standard plug-in card.

#### **Low speed parallel I/O**

A 16-bit-wide input channel and a 16-bit-wide output channel are provided with rates to 1000 words per second. In addition to strobed and handshake protocols, there are modes which provide for output of a predefined data stream at predefined intervals and input of data with associated time tags. Differential TTL drivers and receivers are provided.

#### **Serial I/O**

Serial data at rates up to 10 megabits per second can be accommodated with or without embedded sync patterns. User clocking or internal clocking of data can be accommodated.

Obviously the AIU cannot be directly connected to all the different prototypes that come into the MDSC. However, a level of generality has been designed into the AIU/user interface which permits interfacing most equipment with no additional hardware.

#### *Special test equipment*

The MDSC maintains a set of special test units which are used to verify the operation of the emulator probes. They are used whenever a malfunction cannot be immediately isolated to either the prototype or the emulator hardware. Each test unit is built around a single board microcomputer to which is added a power supply, LED displays and a small amount of additional discrete logic. The function of the box is to verify correct operation of the emulator probe to a certain level of confidence. This is done by executing a ROM resident program which requires correct operation of the emulator address, data, and control lines in order to run successfully. The test units provide a necessary standard of operational reference and act as a test device to verify emulator performance before initial use or at any time thereafter, if a malfunction of the emulator is suspected.

### Costs

The total cost of procured hardware and software for the basic MDSC configuration was \$411,736. This figure includes the cost of the computer and its peripherals emulation hardware, cross software, modems, terminals and PROM programming equipment. It does not include the cost of the AIU, in-house-designed software, and general purpose test equipment. There are now commercially available versions of some of the software that was designed in-house (The Emulator driver and the PROM programmer driver). Purchase of this software would have brought the overall cost up to approximately \$417,000. In its current form the MDSC is supporting over 130 active user accounts, representing 25 different project groups.

### SUMMARY

Given the ever increasing use of microprocessors and the dynamic nature of the state of the art, there is a need to optimize developmental and support strategies so that capital investment, retraining of personnel, and obsolescence of equipment are minimized. Computer facilities have been centralized to advantage in many organizations to provide a range of resources to a large user community. Similarly, development tools for microprocessor support can be centralized to advantage.

Boeing's Microprocessor Design Support Center has proven to be a cost-effective way to provide integrated microprocessor development resources in a large-scale applications environment. The centralized facility resulted in significant savings when compared to the estimated \$649,000 cost of acquiring decentralized stand-alone systems. In addition, there are substantial benefits in efficiency, adapta-

bility to the changing technology, easier and less costly training and, in general, more time available for direct product development.

### REFERENCES

1. Ogden, Carol A., "Microcomputer Support Aids," *Mini-Micro Systems*, February 1978, pp. 35-44.
2. Derman, Samuel, "Low-cost design aids keep pace with growing microprocessor field," *Electronic Design* 25, December 6, 1976, pp. 30-36.
3. "Microcomputers, Part 2: The Development System," *Digital Design*, January 1978, pp. 40-50.
4. Snigier Paul, "Microprocessor development systems - which one is 'best'?" *EDN*, March 5, 1977, pp. 68-78.
5. Clark, Tom, "High End Microcomputer Development Tools," WESCON 77, Session 3, September 19-21.
6. Bass, Charlie and Brown, Dean, "A Perspective on Microcomputer Software," *Proceedings of the IEEE*, Vol. 64, No. 6, June 1976, pp. 905-909.
7. Watson, Irene M., "Comparison of Commercially Available Software Tools for Microprocessor Programming," *Proceedings of the IEEE*, Vol. 64, No. 6, June 1976, pp. 910-920.
8. Clark, Tom, "Troubleshooting Microprocessor-Based Systems," *Digital Design*, February 1978.
9. Francis, Robert, "Real-Time Test Methods for MPU-Based Products," *Electronic Test*, March 1979, pp. 42-46.
10. Francis, Robert and Teitzel, Robin "Real-Time Proto-type Analysis as a Microprocessor Design Aid," *Computer Design*, December 1978, pp. 65-73.
11. Gladstone, Bruce, "Ease painlessly into Microcomputer operation with in-circuit emulation," *EDN*, September 20, 1977 pp. 89-97.
12. Rooney Michael, "Economical Development of Microprocessor Software," *EASCON 77*, 26-1A.
13. Moyer, W. W., "Designing a Microcomputer Test Unit," *Digital Design*, May 1978.
14. 8001 Microprocessor Lab System Users Manual, Part 070- 2464-00, Tektronix, Inc., Box 500, Beaverton, OR 97077.
15. Gladstone, Bruce and Page, Paul, "Distributed processing slashes development system's cost," *Electronics*, August 17, 1978, pp. 89-94 (1979, pp. 32-40).

# Future management concerns regarding office automation

by GARY D. BEAMER

*Pacific Northwest Forest and Range Experiment Station  
U. S. Department of Agriculture, Forest Service  
Portland, Oregon*

The last few years I have become less interested in the vendor shows and I began wondering why? As I thought about going to a show a year ago I went through the usual process of mentally listing my chores and setting priorities. Well, the usual did not happen. I found that the most pressing activity was management awareness. I was already armed with knowledge of tools and correct procedures for procurement, but I had not adequately staffed management. They were not ready to buy. Why was I running off to another show then? I didn't. I set about putting together some ideas to help me reach management, not with the intention of pushing them into an undesirable situation, but searching for a way to gain their understanding. I am still working on it, and here are some of my thoughts.

First let's take a look at the last few years and see what has happened to date. Here are some interesting facts taken from IBM's "Data Processor," September 1979 issue:\*

Twenty-five years ago it cost \$1.26 to do 100,000 multiplications by computer. Today it costs less than a penny. If the cost of other things had gone down the way computing costs have, you'd be able to buy: sirloin steak for about 9¢ a pound, a good suit for \$6.49, a four bedroom house for \$3,500, a standard size car for \$200, an around-the-world airline trip for \$3.

A magnetic bubble memory device had been built by scientists that can store the equivalent of about 100 pages of the Manhattan telephone directory (25 million bits of information) in an area only one inch square. The magnetic bubbles are only a millionth of a meter, or 1/25,000 of an inch in diameter.

In 1953, one million bytes of information could be stored in about 400 cubic feet of space at a cost of \$250,000. An IBM processor can store the same amount of information in 3/100ths of a cubic foot, a space about the same size as a paperback book. The rental cost for the storage is about \$430.

If technology and productivity in other industries had progressed at the same rate as computer-technology, an around-the-world airline flight would take 24 minutes, and a standard size car would get 550 miles per gallon.

Just how fast is fast? Well. . . . .

If you could take a three foot step every nanosecond (billionth

of a second), in one second you could walk around the world 23 times. The IBM 4341 has switching speeds of 3 to 5 nanoseconds, and circuits have been developed that can switch in 13 picoseconds (trillionths of a second).

In one second, an IBM 3033 Attached Processor can execute 5.5 million instructions. In that time, the 3033 AP could receive inquiries from 180 airline reservation clerks, check on whether seats are available and start information back to the clerks. In some 20 years the work computers can do in a second has increased almost 27 times, and the cost per instruction has declined to 1/37th of what it was.

What do we do with all of these fast and powerful things? Do they fit within our needs today? Can we really use these gadgets to help us with our daily chores? How do they fit with our goals? Productivity is a major concern to us all, improving productivity is what these gadgets are all about!

Let's assume we have grown to tolerate these little beasts and have reluctantly agreed that we *need* these new tools to improve our productivity. What will happen in the next decade then? Do we *want* to use them? Will we learn to use them?

The experience we have gained in the last few years working with computer technology has given us new insights. These insights can be described as ideas learned through mistakes and innovations in using word and data processing devices in conjunction with communication devices. This innovative use of word processing (text editing/processing), data processing, and telecommunications networks is bringing about the awareness of "OFFICE INFORMATION SYSTEMS" concepts. Following is a discussion of how I perceive this OIS concept.

The purpose of OIS is threefold: to increase productivity, to provide job enrichment for secretarial, clerical, and technical staffs, and to provide professional staffs with improved information tools and processes (decision support systems).

With the introduction of OIS an organization can:

1. Increase office output with the same staff
2. Maintain previous level of service with reduced staffs
3. Improve control of voluminous data entry, correspondence, and miscellaneous typing and
4. Provide fast turnaround in generating large, complex manuscripts

\* Vol. 22, No. 4, September 1979, published by Data Processing Division, IBM.

Our purpose would be realized if we would: (1) move machines as close to the authors (data and word originators) as possible, (2) merge word processing, data processing, and telecommunications into one activity called, "office information systems," and (3) integrate "office information systems" into the normal work process.

The ultimate, of course, is the integration of computer and communications technologies with "new" management ideas. This means not just automating present processes but using the full range of tools to meet the overall goals of the organization. How to do this is the management concern I am addressing here.

Before we get into this management concern let's talk about some tools. Just what kind of tools or systems could help us improve productivity? There are several vendors who market systems with the ability to do some of the chores mentioned above. Some of these systems are adequate for the learning process, but the greatest flexibility for meeting long term goals will come from maintaining vendor independence as much as possible and thinking broader in concept. Here is an example:

*A local cable bus network:\*\* What is it?*

A local network for interconnecting diverse computers and terminals. The cable bus uses standard CATV coaxial cable and components laid out in a tree-like structure within a building or campus. Equipment at the site would be connected to the network. This equipment could include terminals, plotters, data collection equipment, word processors, the office telephone system, an inter-office video system, and a link to several computers across the United States. Software needed to interface the terminals and other devices will be housed in small, specially designed micro-computers.

*What do you do with a cable bus network?*

- Production typing
- Electronic mail
- Data entry
- Data editing and reformatting
- Sorting
- Mathematical computing
- Programming (applications development)
- Data storage and retrieval
- Text editing
- Data terminal (low and/or high speed)
- Telephone communications

\*\* Wood, David. 1978. Cable Bus Networks for Information Handling. Mitre Corporation, McLean Va. (a working paper.)

Video conferencing, both within a physical structure and to other parts of the organization.

Let us dream for a minute. If we (any organization) install a Cable Bus Network, what might we do with it? We could use it and associated gadgets to arrange travel right at our desk without involving a clerk. We might even eliminate the need for the trip by activating our video conferencing device. We could get information from the same video device on the current activities of any part of the organization. We could check on the most current activities in process in the Senate or House. We could prepare an electronic message for corporate headquarters and have it routed to the appropriate manager for action on the same day. We could prepare a contract for advertisement by modifying a like contract retrieved from our stored library, having it reviewed via electronic mail by all the parties involved, and at the same time have the approvals appended. We could check the status of our budget either by account or in total. I could dream on and on, but with these tools installed and with practice using them, the dream could become a reality.

What about the management concern regarding these devices? With all of these new tools costing less and doing more, it appears that there should be fewer problems. Isn't this going to solve our problems rather than create them?

Pretend for a bit that you are the manager. With the above tools in mind, sit back and think out loud a minute. "We will give the staff some nice gadgets to increase production. We can expect our typing to be perfect and on time, we can have our airline tickets printed at our own office, we can get the AP news at the touch of a button, and we can visit with the branch managers on the video conference system without traveling. Well, there are just all sorts of things we can do!"

Wait a darn minute! What about all of the turnover we have had the last few years? Are we going to be able to find enough people to run these nice, fancy machines? It seems like I hear that swan song every time I visit with a manager. What is the answer? We need to look at the use of this new technology with the intent of creating meaningful work, and we will have to change our way of doing business. That is the "new management" I am referring to. How do I do this you say? We need to implement Carlisle's Office Automation of the fifth kind:†

Here we have the integration of computer and communications technologies with "new management" policy. This is where the payoff is—not in merely "pushing information around faster" but in using the full range of tools to meet the overall goals of the organization.

The situation, as I see it today, is that we are saturated with new tools and really have a problem knowing how and what to do with all the devices available to us. We just haven't learned how to manage them. Here is a quote (by Bill Lippold) that helps explain.

† Dunn, Nina. The Office of the Future. Part I., reprinted from *Computer Decisions*, pp. 16-20,26, July, 1979.

Collecting numbers, running them through a computer, and coming out with the fact that so many machines and so many operators are needed is a mistake. It works in the factory but not in the office culture, which has its own traditions and rules of right and wrong. In the factory people are automated. In the office, it is the principal who needs to be automated. It has to be understood how many dollars in savings are not being realized when the executive does not want to give up a secretary. These subtle attitudes and relationships are not quantified in a computer. Furthermore, studies do not always give an accurate picture of equipment needs. Applications are always different than they are perceived to be, once the equipment is installed, the needs change in accordance with new expectations. Applications come out of the woodwork.<sup>††</sup>

As we encounter a new chore, we add it to our list and attempt to complete it as we have chores in the past. An example: We have a need for more typing, so we get another word processor or typewriter much the same way we did before. We do not sit down and take a broad look at our entire chore list and ask ourselves, "What can we do with our chores that may help our total workload?" "How can we *share* or redistribute them?" We just seem to look at the new job and analyze it out of context. If we take a look at the *whole* list of administrative chores in an organization with a new set of values and at the same time toss out the mind-set we are currently using to evaluate this list, then maybe we could see how to proceed with this new task, using new tools and methods to accomplish it.

#### *How do new values relate to new ways of working?*

Here is an excerpt from an interview by Kristin Anundsen, Editor, *Management Review* with Michael Phillips, business manager, consultant, and author of *Seven Laws of Money*. This article is entitled, "Management in the Briarpatch: An Alternative to the 'SYSTEM'."<sup>\*</sup>

What, exactly is the Briarpatch Society?

It's a community of people who are trying to build a network of new business and work environments that relate to the values of our generation—values that are significantly different from our parents'. These values have to do with learning, sharing, and a belief in "right livelihood." The key is "right livelihood," the concept that there is something unique and special each person can contribute, and that the kind of work people do should relate to these special contributions. These are values that conventional business management hasn't had to deal with thus far but will soon have to face.

Why?

What we used to call the "counterculture" has already made an impact on society. The spread of Eastern disciplines, from yoga to transcendental meditation, is one force that is making radical changes in many people's lives. These people are re-

evaluating their personal goals and ways of living, and they will want new ways of working.

How do new values relate to new ways of working, and to the Briarpatch in particular?

In the Briarpatch, we're experimenting with new management styles. So far, management in the Briarpatch is based on three principles: failing young, learning how the world works, and learning to share. Underlying all three of these is the value of openness. *The Seven Laws of Money* deals with how to be more open about money. Now we're evolving a kind of management that deals with how to be more honest and open with each other.

How is this openness expressed?

Well, for example, the auto repair shops we run have their financial statements posted on the wall, right next to the cost of parts and supplies. Also, the wages of everyone in Briarpatch organizations are public information—in fact, all the information about our corporations is available to all the members and all the customers. We feel we have to be able to justify the wage structure and resource allocation of the companies, to anyone who asks.

Our assumption is that everyone can learn managerial skills if they have enough information and experience. So we freely give each other all the information we can. Since our accounting is posted and open to everyone, the person who makes out a sales receipt, and the customer also, can see how that receipt relates to the final balance sheet.

Let's discuss those three basic principles, beginning with "failing young."

We start with the assumption that failure is desirable, since it's a learning experience. For every "success" in life there are 10 failures—and that means 10 times as many opportunities to learn. If failure is accepted, people are more willing to try. And if they try and fail, they evaluate the consequences of their own behavior more effectively and they understand responsibility better. Of course, when you have the attitude that failure is acceptable—even desirable—fewer things become "failure."

What about "learning how the world works"?

People who are open and growing are looking at all parts of their lives, including work, as a chance to learn about the work around them. Day-to-day business decision making is a great chance to learn, because the consequences of decisions are often very tangible.

To us, the best decision makers are not the ones who come up with most brilliant, rational solutions, but the ones who are able to look at the alternatives and then make "nonrational" judgments, incorporating an understanding that isn't inherent in the situation or the information at hand. The realm of logic and rational thought comprises only 2 percent of "how the world works." If someone makes a successful decision, it's not because he got the 2 percent right, but because he was making some accurate judgment in the remaining 98 percent.

And how do you get to the stage of using that 98 percent effectively?

By sharing—which is the third principle. To share, you have

<sup>††</sup> Lippold, Bill, "Word Processing World," from *Word Processing Systems*, Geyer-McAllister Publications, Inc., 1979

<sup>\*</sup> Excerpted from Anundsen, Kristin (ed.), "Management in the Briarpatch: An Alternative to 'The System,'" *Management Review*, February 1975.

to accept what other people offer you, in a broad sense. When you have really been able to open yourself to other people's experiences and perceptions, you're sharing. It takes time to learn to trust other people's experiences, but when you can you make better decisions.

We have several reasons to take this new look at ourselves: staffing, money, job enrichment, EEO, civil rights, energy conservation, physical space problems, and others. If we integrate these concerns with new technology and new management ideas, we will begin to improve productivity. This seems to be the same as preparing a meal. We take some food and put it together in many different ways to come up with the finished product called a meal. We may use some old familiar items, add to them some new health food supplements, include the use of a microwave oven and have a nutritious meal in less time. All this with less energy and less time. Perhaps, more importantly, the preparation of this meal was a *shared* activity.

Why is it so difficult to look at the use of this new technology in a holistic way? Why is this such a concern? Why is it a problem? Well, change is painful! It is not easy and it takes time, a lot of time and interest. Who has the least amount of time to spend on it? The manager. Who has the most fear about learning how to operate new, complex tools?

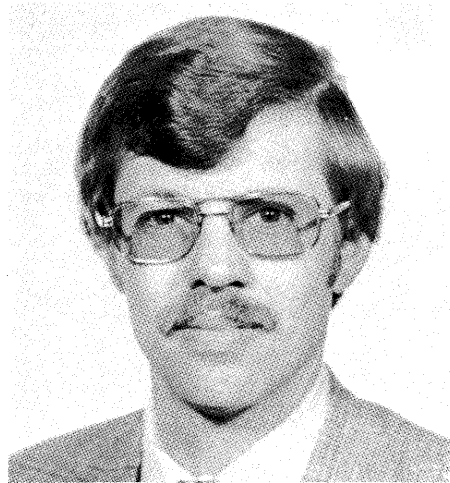
The manager. People who are eager to change are those hungry and in pain, not managers who have reached a level of comfort.

How do we get there from here? How do we do it? Maybe by using a combination of tools that are available today. Some folks are already using some of them, but I'm talking about *integrating* these new gadgets into the *whole* office. Many folks in a given office would use these devices—*not just clerks*. Very simply, chores would be divided differently and each person would have access to appropriate tools to accomplish these tasks. We just don't have time for repetitive typing, rough drafts, and running work through a string of people. This can be started in a rather nonthreatening way by first using people who are interested in new technology and by employee attrition with *astute* recruitment. Once this is established, technology can be introduced where most needed. The key to this concept is not pushing new technology or technological systems or processes. Rather it is encouraging, fostering, and rewarding, any changes in employee behavior that better utilize existing or imminent (potential) technologies, systems, or processes. The focus should be on managers and specialists.

It is important to look at the use of this new technology with the intent of creating meaningful work. We must be conscious of our old values and develop new ones as we learn to integrate these devices into our lives.

## Office Automation

Electronic Mail offers advantages to companies and organizations today and will play an integral part in the development of and evolution toward the "Office of the Future." Four prominent individuals will speak about the state-of-the-art in electronic mail development. A balance of viewpoints will focus on computer message systems, combined voice and text message systems, facsimile, and the distributed electronic mail station. Functionality and cost effectiveness of current systems will be described, and future trends will be discussed.



Walter E. Ulrich  
*Area Director*





# Introduction to electronic mail

by WALTER E. ULRICH

*Walter E. Ulrich Consulting*  
Houston, Texas

## ELECTRONIC MAIL

Electronic Mail is the forwarding of message content by electronic means, and is normally associated with communications between people. In the case of traditional postal service (as opposed to Electronic Mail), the document is physically delivered to its destination using transportation methods. In electronic mail, the content of the message is transformed into electrical signals and forwarded over communication channels for at least some portion of its journey.

Examples of electronic transmission include traditional teletype services, store and forward computer message systems (CMS), and facsimile.

## MOTIVATION

There are two basic motivations for implementing Electronic Mail Systems: 1) to provide a quick, efficient, and inexpensive method for sending and receiving messages; and 2) to increase the productivity of the office (or "knowledge") worker. I will briefly touch upon each one and qualify the market potential.

### *Efficient message systems*

The cost of a first class letter is now 15¢ and is projected to more than double by 1985.<sup>1</sup> Mail service is not universally acclaimed either for its speed or dependability. The cost of a TWX/Telex message is in the \$1.50 price range. A long distance three minute telephone call that is dialed direct from Houston to Los Angeles costs \$1.24 and one from New York to Los Angeles costs \$1.46. A computer message service can now send that same message for between 25¢ and 50¢, and provide some new services as well. An internal CMS can send a message for less than a nickel where there is high message traffic.

Facsimile systems provide for the electronic reproduction of the page at the receiving end. For less than the price of a computer terminal, there are systems that will transmit a page over the telephone network in 2 or 3 minutes. This is especially convenient where graphics are involved or where the sender is transmitting a document that is already physically available.

### *Office automation*

During a recent 10-year period, the productivity of office workers has improved only slightly against nearly a doubling of productivity for the factory worker. The production worker is supported by eight (8) times the capital investment of the clerical employee.<sup>2</sup>

Businesses are becoming more complex and need more and more information. The increasing need for office support, the low level of capital employed to assist the knowledge worker, the increasing wage and salary spiral, and the improving productivity and price performance of computer/communications are together a potent force. These factors justify the implementation and use of sophisticated electronic mail and word processing systems in many organizations today.

### *The market*

Companies see a major opportunity in supplying this market. As cost decreases and functionality increases, Electronic Mail becomes an attractive substitute to traditional methods. Even a small part of 50 million TWX/Telex messages, 60 billion pieces of first class mail, and 250 billion (or more) telephone calls is a lot of traffic.

It is predicted that the number of messages sent via computer message systems will grow from 40 million in 1979 to 275 million in 1982.<sup>3</sup> I believe this projection to be conservative. Over 80,000 facsimile units were to be shipped in 1979, at a value in excess of \$200 million—a 65 percent increase over the previous year.<sup>4</sup> Communicating multi-function terminals and communicating word processors (combined) will grow from some 33,500 units at the end of 1979 to close to 380,000 units in use by year end 1983.<sup>5</sup> While it is difficult to make accurate long term forecasts for such new products, the market potential by any measure is tremendous.

## ELECTRONIC MAIL COMPONENTS

Electronic Mail is a very broad topic which includes many alternatives and approaches. Messages can be text, documents, graphics, voice, or video. Underlying transmission methods range from cable to fibres to satellite to microwave

transmission. Intermediate carriers provide value-added services. User interface methods range from document insertion to dumb keyboards and printers to highly "intelligent" terminals.

This introduction is limited to a historical look at the teletype and an introduction of four aspects of Electronic Mail: computer message systems, a combined message switch/PBX system, facsimile, and a distributed approach to message communications. These four are chosen because they are important, interesting, and representative of the kind of developments taking place in this field.

### *The teletype*

The telegraph was invented by Samuel Morse in 1844, and the teletype was introduced in this country in 1917. Teletype services provide for the entry of message text from a keyboard. A physical circuit is established, and the text is transmitted from the sender to the recipient, typically at 6.6 or 10 characters per second (cps). There are presently some 150,000 or more TWX/Telex terminals in the United States and throughout North America.

Traditional teletype services provide for immediate message delivery. As a byte of information is sent, it is transmitted to the recipient across a physical connection instantaneously. In most cases, using paper tape or a buffered terminal, the message is keyed in first and then sent as a continuous stream of bytes. The device at the far end must be ready to receive the message or it cannot be sent. Furthermore, the only record of the transmission is the billing information and the physical hard copy.

### *Computer message systems (CMS)*

In computer message systems, the message is entered by the sender and stored in digital form by a computer for subsequent delivery to one or more recipients. Because these message systems are computer based, there is an opportunity for providing a wide variety of additional services.

Typical CMS messages are initiated from a keyboard for eventual output on a CRT display or on paper. That is, many of these messages are originated at one terminal for delivery to one or more other terminals. Actually, the messages are first transmitted to the CMS computer(s) and then on to the recipient(s). Typical transmission media include hardwiring (for a local device), leased and dial-up telephone lines, value-added (packet) networks, and teletype networks.

Sophisticated message preparation features might be included. Text editing commands assist the user to modify the message before it is sent.

As text is entered, the incoming bytes are stored in digital form. CMS will maintain a copy of a message as it is being entered into the system. Typically, a copy is also stored in auxiliary storage (disk) at least until its delivery to the recipient(s). On some CMS, that message may be around for a long time.

In some systems, a message can only be sent to one ad-

ressee. Similar restrictions are found with traditional teletype services, facsimile, and the telephone system. Sending the message to multiple recipients requires multiple message entry. Fortunately, this is not the case for most CMS.

In many CMS, there are convenient facilities for sending a message to a small number of individual recipients, to specific groups by using their "group name," or to all users (called a broadcast message). Senders can sometimes generate their own "group name" files, and can mix and match addresses for any particular message.

Filing provisions of CMS sometimes include (1) archival of some or all messages, (2) filing of messages into user-created files (like subject files), and (3) filing of prepared text. In some systems, access to these files can be separately controlled. Retrieval features refer to the user's ability to get at these files and to review and scan the message of interest. For example, a user might want to review all messages on a certain subject sent between certain dates.

Special files might also be maintained. Examples might include an electronic "out basket" for each sender giving the delivery status of all messages sent; an electronic "in basket" giving the recipient a list of all outstanding incoming messages; and a list identifying each special file that a user may have created.

Some systems are location-oriented; that is, the "recipient" is a terminal. This is similar to the teletype network. Other systems are addressee-oriented; that is the recipient is a person or group of people.

Addressee-oriented CMS are classified as "Mailbox" systems. The message, when sent, is flagged by the CMS as "pending" until delivered to the recipient. When the recipient accesses the CMS from any point, the recipient identifies himself to the system. At that point, the message becomes available. In other words, the message is stored in the addressee's electronic "Mailbox" until it is called for.

### *Computerized communication exchange*

Start with a programmable computerized business telephone system (PABX). Such a system might provide some or all of the following features to a business customer: conference calls, call holding, call forwarding, chain calling, answer any station and other night service features, secretarial hunt, paging access, call recording, traffic measuring and storage, optimal and alternate call routing, and lots more.

At the heart of such a system is a computer. A logical growth of this computerized system is to include message switching (or CMS) software as part of the computerized telephone system. Terminals could be conveniently plugged in wherever there are telephone jacks, or even through a plug in the telephone instrument itself. Terminals at remote locations would access the system through the telephone network using a modem or acoustic coupler. The concept of sharing the same facilities for voice and message communications is interesting and can provide certain econ-

omies; the further integration of these communications offers some exciting possibilities.

### *Facsimile*

While there was some earlier work, the "classical" facsimile system was developed by William Sawyer in 1875. Facsimile works by identifying whether each point on the page is dark or light, and transferring the information across a telephone line. The receiving device will either mark the equivalent point on a blank page or leave it white. Since there may be many points per page, depending on the resolution of the unit, a lot of information is sent.

The usefulness of facsimile for business was increased substantially with the introduction of a low-priced, plain paper, desk top facsimile unit in 1966.<sup>6</sup> It interfaced to the telephone network with an acoustic coupler and was relatively easy to use. The transmission would take as long as 6 minutes per page.

Facsimile units are classified on the basis of transmission speed. Group I units, like the one introduced in 1966, are analog devices that transmit a page in 4 to 6 minutes. Group II apparatus employ bandwidth compression techniques to achieve transmission times of 2 to 3 minutes per page. Group III hardware reduces the information transmitted, usually by digital techniques, and sends a page in a minute or less over normal telephone lines.

The tradeoff with facsimile choices are cost versus transmission time and cost versus ease of use. High speed devices are more expensive, but reduce telephone charges and length of operator attention per page. Ease of use features (like automatic page feeders) also add to cost. Key considerations include the volume of documents, location of intended recipients, type of equipment in use by intended recipients (compatibility), and kind of documents.

High bandwidth communication links offer even faster facsimile transmission. To utilize spare capacity on broadband satellite channels, one vendor will be offering a one second (3600 pages/hr) facsimile device. It is envisioned that the documents would be inserted and stored in digital form and forwarded in batches when the links were not being utilized for other forms of communications. Multiple copies would only be transmitted once, and the system would make the added copies.<sup>7</sup>

### DISTRIBUTED APPROACH TO ELECTRONIC MAIL

In early Electronic Mail systems, the user dealt with a simple facsimile transceiver or a dumb terminal. There was no intelligence at the user location. The routing and delivery was handled by the telephone network or a central computer.

The decreasing cost and increasing functionality of computer hardware has made it possible to distribute intelligence to the user. Integral processors make it possible to perform complex compression algorithms in digital facsimile devices. In computer message systems, intelligent terminals make for powerful message stations. Complex text editing features

can be included in the terminal, and messages can be batched or even held until after hours. In both cases, the local processor makes it possible to optimize the utilization of communication bandwidth.

Distributed data processing, however, offers more advantages than just compression schemes and batch message preparation (data entry). In a truly distributed environment, the central mail functions are shared by a hierarchy of processors. Scratch pad notes and drafts are stored at the users mail station. Intraoffice mail is collected at the on-site message exchange, or even shared between the local mail stations. Interoffice mail is handled by regional or centralized message exchanges.

As time goes on, the local message station has evolved from a simple message (data) entry device to a sophisticated word processor. Graphics will be added. A facsimile input device will be included, perhaps with optical character recognition (OCR) ability. Eventually digital voice and freeze-frame video interfaces will further broaden the communication alternatives available locally to the user. Ultimately, why not holographic images providing full color three dimensional face to face meetings—complete in every way except for the handshake.

### ELECTRONIC MAIL: THE STATE OF THE ART

The four speakers at this session will write on and discuss these topics in depth. Their organizations are noted and have been active in one or more aspects of Electronic Mail development.

Computer Corporation of America (CCA) is the designer and developer of the COMET computer message system. COMET is one of the first systems of its kind to be commercially marketed as a stand-alone in-house electronic mail system. CCA has also done some interesting basic research in the Distributed Database area and is active in storage technologies.

ROLM Corporation is a progressive supplier of computerized business telephone systems (PABX), and last year announced its Rolm Electronic Mail System (REMS). REMS is a computer message system that can be integrated into their PABX product. This eliminates the need for local modems and provides for a sharing of voice and message facilities.

Qwip Systems, a division of Exxon Enterprises, has been very effective in marketing its low cost Qwip and Qwip II facsimile units. Funded by Exxon, Qwip has grown rapidly since 1974 and will have an estimated 50,000 units (25 percent of all units) in place by the end of 1979.<sup>8</sup> And they expect to garner an even greater share in 1980.

Datapoint Corporation has been successful in a number of areas related to Electronic Mail including distributed data processing and communicating intelligent terminals. The focus of the company is clearly on the "office of the future." The facilities for Electronic Mail are in place and its importance in tomorrow's office is certain. Datapoint offers a number of computer-based telephone related products as well.

These papers, like Electronic Mail, cover a broad area.

Yet, the fundamental goal is the same: to increase the effectiveness of human and business communications. And these diverse approaches will come together. Electronic Mail is the prototype for and will be the archetypal examples of the convergence of computers and communications.

#### REFERENCES

1. Becky Barna, "Why Electronic Mail Now," *Computer Decisions* (September 1978), 38.
2. L. Duane Kirkpatrick, at The Diebold Research Group: Working Session 78-9.
3. Howard Anderson, "What is Electronic Mail," *Telecommunications* (November 1978), 46.
4. Wayne L. Rhodes, Jr. "Facsimile—New Life for an Old Idea," *Infosystems* (September 1979), 42.
5. Dale Kutnick, "Communicating Word Processors," *Telecommunications* (November 1978), 50.
6. Barry Schreiber, "Facsimile in the Future—Where Does it Fit?" *Telecommunications* (May 1978), 77.
7. Mel Mandell, "One Second Fax via Satellite," *Computer Decisions* (April 1979), 30.
8. Wayne L. Rhodes, "Facsimile—New Life for an Old Idea," *Infosystems* (September 1979), 52.

# Implementation considerations in electronic mail

by WALTER E. ULRICH

Walter E. Ulrich Consulting  
Houston, Texas

## WHAT IS ELECTRONIC MAIL?

Electronic Mail is the transmission of text or a document by electronic means. Examples of Electronic Mail Systems (EMS) include traditional teletype services, facsimile, and computer message systems (CMS).

All such systems must be able to accept input, transmit the message, and output the information. In addition, a number of other facilities might be provided in areas like message preparation, filing and retrieval, addressing and distribution, accounting and control, coordination and content processing, and others.<sup>1</sup> See the companion paper in this proceeding for more information on Electronic Mail features.<sup>2</sup>

## DESIGN CONSIDERATIONS

The selection of transmission media, data base methodology, hardware, and other design criteria are subordinate to meeting the end user requirements. It is necessary to define these objectives first, and then select the right combination of technologies to implement the EMS.

### *Basic objective*

Certainly an EMS must accomplish its mission—the timely delivery of the message to one or more recipients. During research in the general business marketplace the following objectives were identified as general requirements.<sup>3</sup>

1. Low cost—due to the decision process, the justification of such system requires tangible benefits and transmission economies.
2. Reliability—the system must be available when needed, and must deliver the message accurately and dependably.
3. Ease of use—the system must be human engineered with its operator and user in mind, and be responsive in real time to operator action.
4. Control—there must be a provision to monitor and control usage.
5. Terminal independence—compatibility with as broad a range of user interface devices is desirable.

6. Security—privacy and security measures must protect the physical facilities and guard against unauthorized entry or access of the system through communication channels.

Some important characteristics should be also considered during the design phase. The potential for rapid growth in both message volumes and geographic dispersion must be planned for. Furthermore, the system must be flexible enough to change with the dynamic business requirements and growing sophistication of the users.

### *Organizational issues*

In many organizations the impetus for Electronic Mail will originate with the communication's manager. In that case, the system is likely to be a TWX/Telex replacement; it will be designed to minimize common carrier costs. In others, the office services function will assume the responsibility; the goal is likely to be reducing clerical costs and improving secretarial productivity.

In other cases, the management information system function will start the ball rolling. EMS is an entirely different discipline than data processing and needs to be approached with different technologies and orientation. At the same time, a progressive management information systems department that has fully mastered data communications and understands the business needs of the company can leverage on these abilities quite successfully.

The companies with the greatest chance of success have merged these various responsibilities together. Information (computer and office) is a precious asset and communications (data and non-data) is the key to its effective use. Computer and communication techniques of all types are no longer distinct; but are converging. This is a fortunate phenomena offering real benefits, and alert organizations will change to take advantage of it.

In implementing Electronic Mail, it is necessary to analyze the real needs of the organization and its people. Senior management must also be involved.<sup>4</sup> However, I think it is all too easy to get into "the managers must have a terminal" syndrome. That is an extremely complex psychological and motivational area that very few companies are equipped to

tackle now. Good, useful systems can be put together without going that far. The right terminal will sit on the boss's desk soon enough; but take modest steps to get there.

### *Communications media*

The choice of transmission media include hardwiring for local terminals, the dial-up telephone network, leased lines, Dataphone Digital Services (DDS), specialized common carriers and value-added networks (VANs), and shared high bandwidth networks. Except for hardwiring, these alternatives will require either modems or data service units at the CMS and at (or within) the terminal equipment.

Direct Distance Dialing (DDD) provides access everywhere, but is expensive for calls placed during business hours. Wide Area Telephone Service (WATS) can reduce DDD toll charges at certain volume levels. Foreign Exchange and leased lines provide economies between sites that exchange a steady stream of traffic. Compatibility features, code conversion, and error checking must be performed by the CMS and the terminal equipment. The specialized common carriers can provide added transmission economies and some specialized services between certain cities, and some of these are ideal for facsimile users.<sup>5</sup>

Value-added networks, like TELENET and TYMNET, offer a number of useful services. They provide for compatibility for a large number of different terminals. That means, the network will handle the code, transmission speed, and terminal characteristics (for example, ASCII, 300 baud asynchronous, 200 millisecond carriage return delay, full duplex terminal) of a variety of devices and present the data stream in a uniform way to the CMS. They provide error checking within the network (from the node of access to the node of delivery). Perhaps most importantly, VANs provide for the sharing (multiplexing) of higher speed transmission facilities, providing significant economies.

TYMNET, described as a "terminal oriented" network by the inventor,<sup>6</sup> has 450 nodes and is a local telephone call away from 175 cities in the United States,<sup>7</sup> and services 30 foreign locations. TELENET, initially based on ARPANET technology, presently services over 90 cities. Based on the culmination of the GTE merger, TELENET plans to expand to 138 cities, and will implement SDLC in selected locations.<sup>8</sup>

AT&T's Advanced Communication Service (ACS) promised to offer the most extensive range of VAN services if it ever gets off the ground—and there is no technical reason why it should not. While AT&T has recently withdrawn this petition, I believe it will eventually develop and offer a similar service.

In a few years, high bandwidth transmission paths will become available at low costs. Companies with high volume requirements will use these "pipelines" for most communications. One network will carry voice, data, message, and perhaps some form of video communications. In these cases, Electronic Mail should be designed to fit right in.

### *Data structure*

A CMS data base has special properties. One message may be sent to many recipients. The data structure should minimize the number of copies of a message. Pointers to the message contents become associated with senders and recipients to minimize storage and maintenance overhead. The same is true for user names and other data. As functionality increases, each element of data is associated in interlocking ways.

List processing can get pretty complicated even in a simple situation. Let's assume that the system lists, for the sender, mail that has not been read by its recipient(s) in an "out basket," and has an "in basket" so that a recipient can check his incoming mail and choose when to read it. When a user "sends" a message, the sender is linked to the message, and the message is linked to the recipients. All users are linked to their in baskets and out baskets. The message is likewise linked. When a recipient reads the message, for example, the sender is identified. Furthermore, the recipient's in basket and (by following the links) the sender's out basket is updated to reflect the message is read.

Users are sending and receiving messages all the time. There are sure to be conflicts in updating these linked lists. The system must have methods for resolving these conflicts and contention.

Because of the reliability issue, it is essential that all messages are backed up. In a system with moderate functionality, furthermore, the linked lists must be able to be rebuilt. In high volume systems that cannot be taken out of service, the development of a back-up and recovery procedure will compound the complexity of the data base.

Let's add one final, very real consideration that would be of utmost importance in developing a large scale intercompany public service. Because of volume or response or other reasons, it is necessary to distribute the processors and the data base. The question of distributed data base requires a conference of its own. There is a lot of good research being done, but I do not believe that this problem has been satisfactorily resolved, much less applied to a functionally rich CMS.

Data base sophistication compounds exponentially with functionality. For even a moderate system, the data base structure requires extensive list processing, multiple access protection, and sophisticated back-up and recovery methods. Before some systems are fully successful, the distributed data base problems must be solved.

### *Hardware*

Minicomputers should be carefully considered as the delivery vehicle for a store and forward Computer Message System. A CMS is a special purpose system supporting multiple simultaneous users with uneven data rates, experiencing frequent interrupts, and requiring specialized list structures and string manipulation. It has been my experience

that general purpose operating systems and timesharing systems are not well suited for CMS development.

Minicomputer performance makes it practical to develop powerful message systems today. Minicomputer prices justify devoting the computer(s) to that application. Microprocessors, as part of intelligent terminals, make it possible to prepare the text locally; communication line charges can be saved by batching completed messages and transmitting them as a group. Microprocessors will become increasingly important for unloading the minicomputer of specialized communications functions. Ultimately, most functions will be performed by discrete microprocessors organized as modules of a Multi-Processor System.

In the case of facsimile, the basic tradeoffs are hardware cost versus transmission speed. Selected properly, a more expensive unit will reduce communication line charges per page and increase operator productivity.

#### Other issues

There are a number of other problems unique to the CMS environment.

1. Names—The recipient addresses should be easy to remember and appealing to human users. If possible, the users should be able to get the right address without having to search for it. In a large company, try developing a friendly scheme that prevents a message from going to the wrong Dave Smith. And what about the directory problem for Stuart instead of Stewart?<sup>10</sup>
2. Undelivered Messages—Provision must be made for messages that are sent to an addressee, but the addressee chooses not to read the message. What happens when a message is sent, but the recipient's user name is deleted before the recipient reads the message?
3. Interrupted Transmission—When a recipient chooses to "read" a message or group of messages, the CMS will put all the characters in an output buffer and flag the message as having been delivered. When the transmission is through a VAN, the network itself will accept and buffer characters. However, the recipient may be just seeing the first characters at 30 cps. If the transmission is interrupted (line failure, accidental disconnect) the characters that are on the way will be lost.
4. Custom Tailoring—For a large system, some users will need specialized services. Examples might be a special screen format, or a different protocol, or more detailed billing, or customized identification. Certain kinds of "execution files" and user programmed "partitions" might be considered. ACS plans to provide these kinds of options.
5. Legal and Regulatory—There is an extensive body of national and international regulation on the electronic switching of messages. Domestic EMS operated by a company on its own hardware for its own use is unregulated. Otherwise, it is best to check both existing and pending legislation.

## IMPLEMENTATION

Depending on the scope of your EMS, implementing Electronic Mail can be a big job. Careful analysis and planning are an essential prerequisite. A number of specialized technical, organizational, management, and business skills will be brought into play. Good project management and trouble shooting skills are a must. The panelists at this session will offer their own experiences in implementing Electronic Mail.

*Computer Decisions* magazine has editorial offices in Rochelle Park, New Jersey; Sunnyvale, California; Bethesda, Maryland; and Lockport, Illinois. They are experimenting with a well-known vendor supplied computer message service. They are a small organization, and their experience and those of other users is recounted.

Chrysler Corporation has had its share of problems, and has cut expenses in a number of ways. However, a progressive project in Electronic Mail and communications continues to be funded because of important benefits. The status and progress of this project is reported.

Texas Instruments is a company known for technological innovation. Using their own hardware, TI has been aggressively developing and implementing an internal Electronic Mail system. The evolution of this system and its benefits worldwide are discussed.

The airlines have been pioneering users of communication technologies for years—the early work in reservation systems is a well-known example. Texas International Airlines has been increasing its Electronic Mail usage including facsimile, computer message systems, and a post office interface. Their thrust, current status, and future expectations are described.

Electronic Mail is alive and well in the real world. Companies are implementing this technology and achieving the benefits of decreased cost, increased responsiveness, and better utilization of resources. EMS is one of the challenges of the '80s. It greatly enhances the value of the information assets and improves the timeliness of effective decision-making. That is a real competitive advantage, and translates surprisingly soon into improved margins, larger market share, and greater profitability.

## REFERENCES

1. *Electronic Message Systems: The Technological, Market, and Regulatory Prospects*, FCC Contract Number 0236 (Kalba Bowen Associates: April 1978), 14-22.
2. Ulrich, Walter, "An Introduction to Electronic Mail," *NCC'80 Conference Record* (AFIPS: May 1980).
3. Ulrich, Walter, "onTyme, A Computer Message System," *NTC '77 Conference Record* (IEEE: December 1977), 21:5-2,3.
4. Dunn, Nina, "The Office of the Future—Part II," *Computer Decisions* (August 1979), 68.
5. "Electronic Mail: New Transmission Services Help Promote Its Growth," *Communication News*, November 1978.
6. Tymes, LaRoy, "TYMNET—A Terminal Oriented Communication Network," *Spring Joint Computer Conference Proceedings* (AFIPS: May 1971).



7. Harcharik, Bob, "The International Spread of Packet-Switching Networks," *Telecommunications* (September 1979), 103.
8. "Telenet Cites Plans Hinging on Merger," *Computerworld* (April 23, 1979), 38.
9. AT&T, Petition to the Federal Communication Commission, July 10, 1978.
10. Feinler, Elizabeth, "The Identification Data Base in a Networking Environment," *NTC '77 Conference Record*, 21:3.

# Experiences of an electronic mail vendor

by JEFFREY B. HOLDEN

*Computer Corporation of America*  
Cambridge, Massachusetts

Computer Message Systems (CMS) are a relatively new phenomenon (5-6 years) and are the result of work done in the Time Sharing Services industry, computer networks, and Advanced Research Projects Agency R&D activities. Computer Message Systems use the computer as an integral component of human communication.

Using a computer terminal, the CMS performs or aids in message creation and distribution, electronic filing and retrieval, and message reading.

Computer Message Systems are a unique form of electronic mail because their use results in the direct linking of two or more people wishing to communicate. Up till now there were three means of such direct communication: face to face meetings, the mails, and the telephone.

It is interesting to note that two of these so-called "direct communications means" (mail and phones) have been so corrupted in the business world with administrative overhead that it seems rarely plausible to use the term direct (witness the secretary opening/copying and distributing mail and placing phone calls for the manager). Noting this, let's review some specific problems with these three traditional means of direct communication (in particular telephonic communication) and relate them to CMS.

## THE PROBLEM OF LOCATION

If I want to communicate with Mr. X by phone, mail, or meet with him, I have to locate him somewhere on the face of the earth. This is often very hard to do and, as the modern business environment becomes more and more mobile, it becomes harder and harder. Where in the world *is* Mr. X?

If the time is during normal business hours, one assumes he is at his desk. But is he really? A lot of the time, it turns out, he is not. He is in a meeting down the hall, he is in the men's room, he is in transit to another office, or perhaps he is not in the building at all. He may be sick, out to lunch, making a customer call, in a car, an airplane, or the Lord know where.

Furthermore, I may want to talk to Mr. X outside of working hours. This is even harder. He may be at home, at relatives, out to dinner, or at the movies. One thing is for sure. In today's fast-paced world, Mr. X is very hard to find.

## THE PROBLEM OF INTERRUPTION

But our problems have only begun. Suppose I know where Mr. X is, that he has a phone nearby, and that the phone is not busy. What makes anyone think Mr. X will be willing to be interrupted? The chances are that he will not, and I can't say I blame him. I do the same. A lot of the time I'm in a meeting, because I'm doing some work that I would like to continue doing. (Of course, for me even to say I'm in a meeting requires an interruption in my work.)

So what happens? I call Mr. X and leave word. He calls back and leaves word. I call him back and leave word. He calls back, etc. (This game is called Telephone Tag.) I have known cases where this has literally gone on for weeks. By the time I got through to the other person, I had forgotten what I wanted to tell him.

Let me give you some statistics of my own use of the phone. I did a study and discovered that, of the calls I placed, only 26 percent of them went through on the first try. This means that on the average I have to place almost four calls in order to get a single one completed successfully.

What were the problems? All kinds. In 38 percent of the unsuccessful cases, the person being called refused to be interrupted. In another 38 percent of the cases, the number didn't answer (most of these were internal calls). In 14 percent of the cases, the called number was busy, and the remaining 10 percent represent miscellaneous problems such as the line being lost before the called party answered.

Altogether, a great deal of aggravation.

## THE PROBLEM OF TIME ZONES

So far we have been assuming we are communicating within the same time zone. But what if we're on the East Coast calling the West Coast? Let's assume that business executives work 9 to 12 and 1 to 5, five days per week. That means that at best, a manager is in his office 35 hours per week. But if two managers are on opposite coasts, they are simultaneously in their offices, at best, only 15 hours per week. As we have seen, it's hard enough to reach anyone on the phone given 35 hours to try in. When the window is reduced to 15 hours, the problem is roughly doubled.

And what happens if we're in New York trying to communicate with Tokyo? Now the "telephone window" has shrunk to zero. There is simply *no time at all* during working hours that one manager can hope to reach another.

### THE PROBLEM OF RECORDS

When you use the phone, there is no record of who said what to whom when. For many business purposes, this makes the call virtually useless. In my own case, I have long ago gotten into the habit, after all but the most trivial calls, of picking up my dictating machine and dictating the substance of the call. The dictation belt goes to my secretary, who types it out, and eventually sends me the typescript. I then scan the typescript, make corrections if necessary, and put it in for filing.

In short, one phone call generates a dictation task and an editing task for me, as well as a typing task, a duplication task, and a filing task for my secretary.

### THE ONE-TO-MANY PROBLEM

In the business world, one person often wants to communicate with many people. Yes, it is possible to set up a conference call on the phone. But it's so hard to set up and it's so unsatisfactory that it's very rarely done in practice. Typically, if a business man wants to communicate with a group, he gives up on the phone and has a meeting or writes a memo.

### THE PROBLEM OF INFORMATION DENSITY

The phone shares a problem with all speech communication: the information density of speech is very low. Generally, the electronic transmission of speech requires about 60,000 bits per second. These 60,000 bits of speech carry about the same information as 15 characters of written text. (Try it—in one second you can read out loud a passage of about 15 characters).

But you can transmit 15 characters directly as text by transmitting only 120 bits of information, rather than 60,000 bits of speech. If you insist on transmitting speech you are transmitting 500 times too many bits. And all these bits have to be paid for. In a very fundamental sense, speech is an uneconomic medium of communication.

### THE PROBLEM OF LONG-WINDEDNESS

My final problem is that the conventions of our society require us to be long-winded on the phone. One must inquire about the other person's health, or the health of his family. How're the kids, George? There is the obligatory discussion of meteorological conditions. Pretty chilly out today, wouldn't you say? When I measure the length of my own phone calls, I was surprised to find that my average call took 4.8 minutes.

It is almost impossible to get someone on the phone and

say, "This is Jeff, your plan is approved," and hang up. That would only take 3 seconds instead of 4.8 minutes. But our social conventions won't allow it.

### THE COMPUTER MESSAGE SYSTEMS

Let's now turn our attention to CMS. CMS do not require you to locate anyone. They never interrupt. Time zones don't matter. All communications are automatically recorded and filed. One message can go to multiple recipients. Computer Message Systems are based on transmission of text, which has high information density, rather than transmission of speech, which has low information density. Messages are short rather than long-winded.

In fact, now I believe one can appreciate this definition of CMS. "Computer Message Systems are a means to communicate and record communication in a timely manner without locating or interrupting the recipient and without undue administration."

I concur with some "experts" who predict a doubling of the CMS business in the U.S. over the next 3 years. During that time, most major corporations and government agencies will have accomplished some pilot evaluations of CMS and some will even begin full scale implementation. Interestingly, the real movers in this market may be in the secondary tier organizations where a bolder attitude prevails.

In the longer term, CMS may earn their places as an assumed means of managerial communication, once again establishing control of person to person communication in the rightful place of the individual doing the communicating.

Prior to this happening, though, there is the necessity to address certain real or imagined problems with CMS.

#### *The organizational concern*

The first of these problems is how does the Computer Message System fit in the organization. Or, who is in charge here?

This, of course, leads us to the debating candidates.

MIS—Telecommunication services—and administrative services. Certainly there are strong cases for all three but in the end I doubt that it really matters. What does matter is a corporate level of commitment (in terms of funds and moral support) to whoever is assigned the responsibility. Beyond this is the overpowering requirement that CMS be understood as a service entity allotted nearly no margin of error. The key to success is response to demand. A rule to heed is that the Computer Message System is only as good to the user as his last experience with it. Note that the lack of this response to demand has been the very downfall of the mails such that today nothing important happens by way of the mail. Would you bet your job on a USPS delivery?

#### *The cost concern*

A second immediate problem we are all faced with is the cost justification hurdle. Anytime anybody needs to kill anything this is the mode of attack.

Before venturing on a path to establish an air tight cost justification model for CMS, I maintain it is important to establish relative affordability.

So let's compare some unit costs:

First, let's compare the COMET Computer Message System and the telephone. A three minute telephone call from Boston to L.A. during business hours costs \$2.44 plus tax, while a measured 16-line message to be composed, edited, filed, transmitted—and in L.A. read and filed—costs \$1.07 using the COMET Service Rates.

Next, let's compare COMET and a memo or letter. According to Dartnell Institute, the cost of producing a single business letter is \$4.47; others say the cost is as much as \$18.00.

Finally, let's compare COMET and TWX. A recent review of a company's TWX service indicates that what costs \$20,000 a month in TWX services would cost only \$13,000 using COMET.

These cost comparisons provide proof of affordability; however, it should be noted that when one considers CMS value added services, the case becomes even clearer. Still, we have to go beyond merely comparing costs and, for this, let me suggest various "Justification Scenarios" which may serve to be more important as they point to individual productivity improvements.

One is span of control, the idea being that the number of managers or supervisors could be reduced and the work remain constant. A second area involves reducing the extent of interruption thereby increasing the amount and value of work. Another scenario could be based on reducing supporting shadow functions around communications. An opportunity in some applications centers around the speed with which information is transferred. Finally, but perhaps most importantly, is *time savings*.

Let us use the last of these (time savings) and follow the scenario for possible cost justification.

In my view, the primary cost benefit of Electronic Mail is in the executive time which is saved. What is the cost of managerial time? Take a \$50,000 a year manager. Add 30 percent overhead and assume he worked 1800 hours per year. It then turns out he costs his employer \$0.60 per minute, or just one cent per second.

What are the costs of Electronic Mail? You can subscribe to an Electronic Mail Service (use of a time-shared central computer that runs the Electronic Mail program) for \$60 per month, and you can rent a terminal for \$90 per month. Hence, you are in business for \$150 per month per subscriber. If a company buys an in-house Electronic Mail system, including terminals, and shares the terminals among a reasonable number of people, the cost can drop as low as \$20 per month per subscriber.

But let us use \$150 as an upper bound. Now, if our hypothetical executive can save just 12.5 minutes per working day through the use of Electronic Mail, he will pay for his use of the Electronic Mail service. If we are talking of an in-house system, 12.5 minutes per day will pay for his use of the system many times over.

In fact, it is my impression that an Electronic Mail system

saves a manager not merely 12.5 minutes per day, but many times that amount. Take my own case. On an average working day I deal with 24 messages (I receive 14 and send 10). The time spent in doing that is 16.4 minutes. If, instead of using Electronic Mail, I used the phone for these 24 messages, considering that my average phone call takes 4.8 minutes, I would be spending 1.9 hours on the phone. I would therefore waste about 1.6 hours per day. For our hypothetical manager, this would cost \$58 per working day. Over the course of a month, he would recover the cost of his use of an Electronic Mail service seven times over. If we are talking of an in-house system, he would recover the cost fifty times over.

(The costs of Electronic Mail discussed above have not included toll charges for telecommunications—Telenet and Tymnet. In comparing the cost of Electronic Mail with the cost of the phone, we can consider that toll charges are roughly equal in the two cases. If anything, since Electronic Mail interactions are so much shorter than phone calls, the comparison would probably widen the gap in favor of Electronic Mail.

It should be pointed out that our cost analysis so far has taken account only of the manager's time in reading and writing messages as compared with talking on the phone. It has given no weight to the fact that, on the average, each phone call has to be placed four times, to the fact that if the executive wishes to have a record of the phone call he has to dictate or write it out, to the fact that Electronic Mail does not disrupt him many times a day, to the fact that he has no time zone problem, etc. If we took these additional matters into consideration, the cost advantage would be even greater.

But ultimately, cost savings may not be the real point. Perhaps the key is that the typical manager is overworked, always short of time, and constantly hassled. Electronic Mail provides relief. It makes him more efficient by organizing his communications and allowing him to be master of his own time. I would be very surprised if Electronic Mail did not become the communication standard for the business person in the next decade. It's simply a better way to live.

#### *The human behavior concern*

A third issue of present concern is that of human behavior. Past experience has taught us a lot. Although there is plenty of room for improvement, many vendors are skilled at employing human engineering and growing numbers of users effect change through understanding and involvement.

Some particular human behavior problems CMS encounter are: the satisfaction curve, command language and typing.

The satisfaction dip, also termed buyer remorse, occurs when there is initial excitement surrounding this brand new thing, followed by a realization of the limitations of the system (disappointment) and, finally, a rise to a stable realistic satisfaction level.

The user command language problem is the responsibility of the vendor or designer. Failure to underestimate the needs

for simplicity of language, friendly response, and natural or expected flow surely dooms the CMS.

Finally, is there a problem in typing? Surprisingly little. One executive states he would be absolutely incapable of typing a business letter, but has no trouble with Electronic Mail. Why is that? I think there are three reasons. First, the messages are short (if I want to send a long one, I ask my secretary to type it for me from her terminal. But for ordinary messages, it's much quicker for me to do it myself). Second, the system helps by providing editing facilities that make it easy to correct errors. Third, for some reason that I don't fully understand, it doesn't bother me to send out a message with a couple of typos. (By contrast, I would not tolerate a memo to go out over my name with even a single error.) Evidently the psychology of the Electronic Mailer user makes him very relaxed about such cosmetic issues.

### *The technology concern*

The final issue of concern I will address is that of the technology necessary to support CMS. Many of the technological pieces are obviously ready (witness progress in the terminal arena, packet networks, and the general cost performance trends of hardware). But underlying these obvious accomplishments is the realization that providing Computer Message System capabilities to a group of 1000 is one thing; for 100,000 quite another. At the latter user population level, the true technical challenges surface and they are concentrated in solving traditional data base problems.

As an illustration of the type of problems, let us note that when the user population exceeds 100,000 names, the probability of a name ambiguity for an addressee is over 70 percent. In contrast, below 1,000 addressees, the problem hardly exists.

To provide CMS for these large populations requires sophisticated new software techniques applied to the areas of distributed data bases which afford reliability, response, and reduced communication costs.

And, of course, the thousands of users of a CMS will be generating thousands of messages for storage and retrieval. It is encouraging to note that these supporting technological building blocks (that is, a distributed data base system and large scale storage and retrieval capacity) have been accomplished on the ARPANET System. The latter is a system called Datacomputer which encompasses 3.2 trillion bits of storage (this is equivalent to 1500 IBM 3350's) and allows users to store and retrieve messages by any word or combination in the header or text. The former is a system entitled SDD-1, which is the first working distributed data base management system in the world.

### CMS SUCCESS STORIES

Nevertheless, CMS is beginning to make its mark, and to illustrate this, I will outline some CMS user experiences and glean sensitive factors for implementation consideration.

The first case is a Fortune 100 company that produces minicomputer systems and peripherals. They have an in-

house Computer Message System that consists of a PDP 11/70 with 300 megabytes of on-line storage supporting over 700 users via 29 lines. For historical background; this Computer Message System has been in operation since January of 1978. For the active users of this system, the average number of daily logins is three, while average daily time logged in is 25 minutes. The user population profile is heavily weighted toward managers and professionals. Use of this Computer Message System includes broadcasting of information, information inquiry/response, task assignments, follow-up on task assignments, requests for action, status reports, meeting agendas and/or minutes, follow-up on conversations, and informal discussion of issues. As a result of its use there has been a decrease in the number of phone calls as well as the number of interoffice memos, while the number of meetings remained the same. Finally, the users have noted a productivity increase.

Our second success story is a Fortune 100 conglomerate in the communication and electronics industry using CCA's COMET time-shared service. This system consists of backed-up 11/40's with 250 megabytes of on-line storage. This account has been active since April, 1979. It places the number of subscribers at 150 while the activity level is again placed at 2-3 logins per user per day. The user profile chart indicates that managers and executives comprise 75 percent of the total user population, with salesmen at 10 percent, technicians at 5 percent, and office and clerical workers at 10 percent. The uses of this Computer Message System include broadcasting of information, information inquiry/response, task assignments, follow-up on task assignments, requests for action, status reports, meeting agendas and/or minutes, follow-up on conversations, and informal discussion of issues. The results of its use show, once again, a decrease in the number of phone calls as well as the number of interoffice memos, while the number of meetings remained the same, and the users' productivity increased.

A third success story is that of a multinational oil firm also using the COMET service. The account history indicates that the Computer Message System has been in use in this area since March, 1978 with the number of subscribers at 50 and the activity level at 1-2 logins per users per day. The user profile, again, is weighted heavily in favor of managers and technicians, with managers at 43 percent, engineers at 4 percent, technicians at 40 percent, and office and clerical workers at 13 percent. The account applications for the Computer Message System have been in personnel (labor negotiations), finance, project control and inventory control. The results are consistent with a decrease in the number of phone calls, a decrease in the number of interoffice memos, with the number of meetings remaining the same and user productivity increasing.

Common characteristics of all these success stories are: a high level of management use and support—the CMS is solving a real communication need, a critical mass has been achieved, and there has been a reasonable time of experience.

The bottom line lessons for anyone implementing CMS are to obtain top level buy-in, use a real application, use the complete application (mass), and give it enough time.

In conclusion I will leave you with this. You have seen and heard and discussed Electronic Mail (EM), and I hope we can keep focused on the importance of all this. It is important for us as a nation in the face of a lagging economy and this fact is centered around the need for office productivity improvement. It is important for your organization because EM will allow it to run better and leaner, capture

more market share, run higher profits, hire more capable people. This is especially true for those organizations that recognize the opportunity and seize it.

Finally, it is important for you and me because a more successful economy means better more plentiful goods, and improved company performance means better pay and benefits. Do not underestimate the value Electronic Mail can play in your life.



# Electronic message system as a function in the integrated electronic office

by HAROLD E. O'KELLEY

*Datapoint Corporation*  
San Antonio, Texas

## INTRODUCTION

The fully integrated electronic office is composed of six fundamental elements, or functions:

- Dispersed Data Processing
- Communications Management
- Word Processing
- Electronic Message System
- Individual Computing
- Information Storage and Retrieval

In order to evaluate trends in what some industry people call "electronic mail" it is necessary to first define each of the functions of the evolving electronic office, of which handling internal messages is only one factor. For too long we have approached problems in the office on a piecemeal basis, i.e., solving problems one at a time, and independently of the other.

Technology is furnishing totally new concepts for problem solving and we see a rise in thinking at a systems level, rather than thinking in terms of independent machines or functions.

### *Dispersed data processing*

In its simplest form, dispersed data processing is hardware and software that allow the local user, no matter what his organizational level, to program his own machine. That is the key: He can program his own machine. This then implies that the user is free from the traditional centralized host processing environment. Yet, even with that freedom he maintains the ability to communicate with other computers. The dispersed computer can operate—besides as a stand-alone machine—as part of an integral network as a host to even smaller dispersed equipment and terminals.

An important factor, the software technology—the ability to have an easy-to-use, programmable system—is a key element in labeling a product as a distributed processing product. The distribution of dispersed processing equipment and its utility in the electronic office can be either functional or

geographic. Communications, therefore, is a necessary integral part of distributed processing and office functions.

Traditionally, the functions of early dispersed data processing were primarily (1) intelligent data entry or the very efficient single-point of entry data capture; (2) batch processing of that data and the preparation of local reports; (3) shared processor, the clustered terminal approach or business time sharing; and (4) interactive processing. Of course, (5) stand-alone processing includes both telecom to the host or stand-alone without telecom to the host. As we move to advanced dispersed data processing, we are using a general purpose computer that does all of the above principally through software changes.

Dispersed data processing has provided us with these small in size, but powerful, computers. We can apply the same techniques used in DDP multi-function systems.

IBM has recognized that dispersed or distributed processing is important in the commercial end-user environment and has endorsed the concept. So every other manufacturer of small computers is making a valid attempt to relate his product line to this environment to handle office functions.

### *Communications management*

Over the years, most large organizations have built up larger, more comprehensive telecommunications systems. These systems have grown because management realizes that good telephone communications are an essential element for the smooth functioning of their businesses. But as these systems have grown, so have the costs of telecommunications services. Higher rates, especially the frequent increases in long distance telephone rates, have raised telephone costs to one of the largest items in the corporate budget.

But while telecommunications costs have increased, the ability of management to control this expense has not kept pace. The heart of this problem is that the basic information to make intelligent management decisions is just unavailable. Telephone systems, whether telephone company-operated or privately owned, have not included features that provide information for telephone management and control.



Today, newer computer-controlled telephone systems provide many new features and services, but there is still little product innovation and far too few alternatives from the telephone company for management to take effective action to solve telecommunications problems.

The same computers that drive DDP systems can be harnessed to control voice telecommunications. Software changes leverage hardware investment into a truly multi-functional system. Management has additional new tools as additional functional concepts are developed and systems design brings more total integration.

For example, the same computer engine from a dispersed processing system powers a Long Distance Control System for control and management of outbound long-distance telephone communications. Only an intelligent switching subsystem and turn-key software are added to produce a system that requires no user programming and can be utilized with any standard PBX or Centrex telephone system, to control DDD and all types of WATS, Foreign Exchange, Tie Lines and other telephone facilities and optimize call placement. Even remote locations can be centrally controlled with the same efficiency.

In urban areas with local message-unit charges, a similar system can also handle local traffic with complete user charge-back capability, and becomes a Station Message Detail Recorder.

For incoming calls, the same engine again drives a fully-featured Automatic Call Distributor for uniform distribution of incoming calls into agent groups such as reservation or claims services, classified ad placement, order taking, etc.

With the commonality of processors, peripherals and software, these communications management and dispersed data processing applications can be handled with ease.

### *Word processing*

Word processing today is generally performed either at a clerk's desk in a compact stand-alone electronic typewriter or CRT with memory, or in a pooled location where multiple electronic typewriters or CRTs share a larger memory facility and perhaps a higher quality/speed printer.

Now, if the same general purpose computer that powers the other office functions and systems also drives the word processor, we can begin to share and exchange files and use the communications facility to handle the intra-company transfer of files, data, correspondence, messages, etc. The personnel data base can supply data on people, locations, departments to add to the communications data base for telephone numbers. With data files, word processing, and communications management we have the beginnings of handling internal mail electronically and automatically, all within the same system.

### *Electronic message system*

In the fully integrated electronic office, if you have data processing, control of the telephone system for both voice

and data, and word processing, then the mail or message function becomes almost a technology by-product. All the elements are there; why not implement it?

The optimum advantage this level of systems integration has is that data need be input into the system only once . . . at any one of the multi-function stations—and it becomes a resource to the total system with no further intervention. The output of any one functional part becomes the input to any and all others. All functional parts of the integrated system "talk" to all other functional parts, yet can "sign-off" and be independent at the user's command.

### *Individual computing*

Individual computing is the placing of the full power of a company's computer and all information in its common data base at the fingertips of every executive, administrator and clerical worker in its offices, with necessary security protections, of course. This, too, is a by-product of the fully integrated electronic office.

Each individual's work station becomes his personal access to the system for sending and receiving messages, researching and evaluating data, and a myriad of other systems functions.

### *Information storage and retrieval*

Information storage and retrieval is inherently part of all the other functions of the fully integrated electronic office. This function manages the files and serves as the intelligent library resource for the entire system.

### *Utopian?*

Perhaps all this discussion of the logical way the electronic office can be assembled sounds utopian. It is not. This system exists in its fully integrated form and its unique architecture has been proven over years of research, development and in-place commercial use.

The technology and architecture that permits full integration of disparate functions into a common system is called attached processing. Attached processing offers as its basic premise the idea that a computer system can be designed to accommodate the specific and varied needs of a business rather than the business tailor its demands to the requirements of the computer. The tangible product of the attached processing concept is Datapoint's ARC™ System (The Attached Resource Computer). This is an extremely efficient and adaptable, though totally integrated, computing facility which links together an arbitrary number of functionally dispersed smaller computers by means of a high-speed electronic pathway, or bus, and a fully compatible library of systems software. One major difference then in what we have been doing in the past, which may be considered geographically dispersed data processing, is the arrival of true *functionally* dispersed processing.

Functionally dispersed as it is, each user of the ARC sys-

tem has complete and immediate access to all system components—the data processing units, common data base facilities, and various peripheral devices—no matter where they may be physically located in the system. By the same token, the ARC system can sustain many different types of applications—data entry, batch and transaction processing, data base inquiry, data communications, and of course, word processing and electronic message functions—in the most efficient manner possible. A wide variety of business tasks can be performed simultaneously on the same computer system using the common resources of the system with no one user bound by the activities of another.

An attractive feature, and key element of the ARC system, is its modular architecture. This system can grow both in terms of power and the task to be performed as the business it's serving grows. Whenever more processing power or faster data access times are necessary, an additional processor or two may be easily attached to the existing system. Should additional data storage space be called for, more disk drives can be attached to the common system data base. Impressively, this sort of system reconfiguration and expansion can be accomplished while the system remains in operation and does not necessitate changes in the existing application programs for operating systems software.

The Datapoint ARC system avoids many of the pitfalls that are normally associated with more conventional computer systems. Unlike the traditional computer architecture, the ARC system does not require re-programming or computer-upgrade investments each time more processing power or a larger database is called for. Likewise, the ARC system, capable of supporting a variety of functionally dispersed tasks with one common database, is not dependent upon the relatively slow telephone communications or physical media transfers that are normally used to link separate databases in multiple-computer configurations. Rather, the transfer of data is accomplished at extremely high speeds over the ARC system interprocessor bus and is always completely transparent to system users.

ARC systems, whether small or large, incorporate three basic components: applications processors, file processors, and an interprocessor bus. Applications processors, an almost unlimited number of which may be contained within the system, are dedicated to performing batch or transaction processing tasks either in single or multi-user modes. Freed from the time-consuming data storage and retrieval tasks, these processors operate at extremely high speeds to get more actual data entry and data processing work done. File processors, on the other hand, are dedicated to the management of data and data storage units. Because this is their only task, they can locate and deliver remotely stored data to the applications processors as fast or faster through software techniques than this data could be retrieved from local disk storage areas.

The ARC system interprocessor bus includes a number of hardware and software components, all of which are used to connect applications processors and file processors into one totally integrated computer system. An essential hardware component of the system interprocessor bus is the in-

expensive coaxial cable which physically connects all the other components of the bus. Another component of the interprocessor bus is the Resource Interface Module, or RIM, a special purpose data transfer module which connects directly to the processor input and output bus. The RIM provides a unique address for the processor and ARC system and allows data to be transferred over the system bus at exceptionally high speeds. The ARC system RIMs, in turn, are linked to the system interprocessor bus by means of passive or active hubs. A total of 256 processors can join a single ARC node.

ARC systems may be comprised of as many or as few resource units, processors and peripherals as a business requires, and any number of possible configurations. This is possible because the growth and the shape of the ARC system is determined solely on the basis of each company's own functional requirements and not on the basis of conventional computer architecture. Since the ARC system architecture does not employ a central controlling host computer, failure of any individual processor in the system will not bring all operations to a halt.

Should an ARC system unit have to be taken off-line, operations will continue without interruption and all other system components function just as before. If a file processor goes down, for instance, its disk units can be transferred readily to another file processor or even to an application processor that has ample data storage capabilities. Applications processors, printers, and other system peripherals can also be interchanged quite easily in the event of a failure.

Should the need arise, access to data in the ARC system can be restricted under several types of security controls. For example, with the built-in security provisions of the ARC system, any user may designate portions of the common database as "Restricted." Data can also be restricted by locally attaching disk volumes to a systems applications processor. These directly attached disk drives are completely private and can be accessed only by the applications processor to which they are attached.

Another important feature of the ARC system architecture is the optional capability it provides to interface to large central mainframes. By means of the Datapoint Direct Channel Interface Option (DCIO), an IBM 360 or 370 is permitted to participate in the ARC system by attaching to the interprocessor bus. Acting as any other applications processor within the ARC environment, the IBM 370 can utilize data stored in the common database to execute a variety of mainframe application programs running in any language. The most recently announced feature of the ARC system includes the addition of the INFOSWITCH communications management system to manage voice communications on the ARC system. Truly, then, this is the realization, the merging of disparate functions.

I think we all agree that the market direction in the 1980s is to the electronic office. However, the office is the only business function that has not been significantly affected by automation and it will require a great deal of technology and an attention to human interface to mask that technology to achieve market acceptance.

Some of the technological trends which we have observed are the convergence of computer and communications technology. As we demonstrated in our ARC system, data processing, voice communications and data communications are available today in a single computer system. Continued evolution of semiconductor technology, microprocessor, LSI and VSLI devices will offer cost improvements. Lasers will have a major impact on printing and imaging. Fiber optics will help reduce the cost of terminal and computer interfaces. Computer graphics will be in evidence as the requirements for sophisticated output as MIS evolve. Of course, underlying all these hardware technologies will be increasingly sophisticated software and programming techniques. The re-

sult, then, is that over the next few decades the office of the future will become the office of today, or the fully integrated electronic office.

Electronic mail, or electronic message systems, will require an inordinate amount of overhead if they are installed as stand-alone functions. The other option is that they only do a portion of the function, with no pretense of integration into the total office. I believe, however, that through proper integration of these office functions into a common system—electronic message systems included—the maximum efficiencies, effectiveness, flexibilities and productivity gains will be realized.

# The growing use of electronic mail by airlines

by JAMES C. GOODLETT

*Texas International Airlines*  
Houston, Texas

## INTRODUCTION

Electronic mail is one of several computer-based applications which is rapidly becoming a fundamental part of business life. Technology has supplied the data processing and communications industries with the necessary tools to enable them to make the transition from the mailman to direct electronic message delivery.

"Electronic mail is defined as person-to-person communication of messages, using electronic means for capture, transmission and delivery of information. The information is communicated visually, including text and graphics—all the forms of information that can be communicated in letters via physical mail services. The messages can be displayed on a screen or a hard copy form can be generated."<sup>1</sup> As noted from the above definition, the scope of electronic mail is very broad; however, even this definition is not encompassing enough to include the automatic generation of messages being triggered by computer functions.

This paper describes several uses of electronic mail in the airline industry as an example of the importance and diversification of its employment. Likely hindrances to further advances are discussed along with some projection for the future.

## HISTORY

The nature of an airline operation requires the rapid movement of considerable quantities and diverse forms of information. An airline passenger cannot only book space on a flight, he can also call for special food, special services like wheelchairs, book space in a hotel of his choice, reserve a rental car and request that his ticket be sent to him in the mail or available at the airport at a convenient self-ticketing device. The reservation can be for the airline called or for space on a combination of airlines. Each of these special services requires messages to be sent for confirmation and status.

The operation of a flight also involves considerable coordination and message movement. Flight crews must have numerous pieces of information including: (a) a release from a FAA Certified dispatcher, (b) a flight plan showing route, fuel, aircraft type, and alternate airports, (c) a takeoff analysis for the specific flight and load, and (d) weather infor-

mation covering the origin and destination stations and the path in between. Station personnel gather, coordinate, and disseminate various necessary data relative to a flight such as: (a) aircraft time of arrival and departure, (b) passenger and freight data, and (c) weight and balance calculations.

Due to the need for current information, airlines have traditionally been heavy users of voice telephone services. These telephone services, however, did not capture the information for historical or analytical purposes. They generally lacked timeliness and proved to be inefficient as the message still had to be written down and usually copied for use by crew, station and General Office personnel.

Similarly, messages regarding needed aircraft repairs (called "squawks") along with requests for and movement of materials and tools were handled by phone with similar results. With the rapidly increasing size and complexity of airline fleets, the FAA instituted requirements for accurate historical records of all actions relative to the operation of an aircraft.

The first computerized airline reservations systems became operational in the late 1950's. By the mid 1960's several airlines developed stored message or "Passenger Name Record" systems. The most widely acclaimed of these was the American Airlines "SABRE" system employing state of the art IBM 7090 computers and the telecommunications networks using the IBM 1006 Interchange discipline which became known as the "SABRE" code.\* Delta Airlines' DELTAMATIC and Pan American Airlines' PANAMAC Systems followed soon afterward. These systems and their successors (primarily the IBM Programmed Airlines Reservations System or PARS) were important in that they required message switching capabilities to service the quantities and diverse locations required. To service the message delivery need, the airlines used either the simplistic message switching system in PARS or developed independent systems such as the Univac 1108 system at United, the Univac 494 systems at Eastern and Northwest Orient, and the GE Datanet 30 system at Braniff. Essentially all the airlines supplemented these systems with teletype-oriented systems for operational traffic, servicing maintenance and flight operations.

\* This code, also known as the Airline Line Code (ALC), is a 6-bit code, using a synchronous discipline with dual synchronizing characters at the start followed by addressing information, a variable length data field, an End of Message character and a Cyclic Check Character. This code is used by all but two of the major U.S. Scheduled Carriers and numerous international carriers.

Some of the above messages were originally handled through company pouches. Technically, physical delivery was possible for all correspondence; however, to be effectively utilized, information about passenger and aircraft movement required much more rapid delivery. The use of 110 Baud, 5 level Baudot codes spread throughout the industry and TWX and TELEX were extensively used between airlines. Papertape transmission and capture was used for the inter-station and inter-airline movement of data.

With the massive expansion in leased line networks and the high costs associated with these services, the United States Airlines formed a nonprofit organization, Aeronautical Radio, Inc. (ARINC)\*\* to lease circuits at quantity discounts from the phone companies. Along with considerable demand for data services, the airlines are very large users of circuits for voice transmission principally servicing their reservations offices. The domestic carriers as a whole are second only to the U.S. Military in the use of common carrier communications services. Individual carriers like United and American with over 500,000 circuit miles each of leased lines for both voice and data would spend an estimated 46 percent more without the discounts available through ARINC.

The following section gives a sampling of several areas of current employment of electronic mail. The generation and transmission of messages has been an integral part of most real-time systems developed over the last decade to service such departments as Flight Operations and Maintenance and Engineering. A clear differentiation has been difficult between "mail," whether manual or electronic, and the message associated with computer stored on-line data bases with real-time access. Somewhat surprisingly the Flight Operating Systems and the Maintenance and Engineering Systems built by the airlines are amazingly diverse from an equipment and network standpoint. Functionally they are very similar especially in their need for message delivery services.

The prime purpose of a Flight Operating System is to track and control the movement of aircraft. A FAA licensed dispatcher is responsible for developing key information relative to the flight of a plane between two cities. He must also communicate this information to the station, flight personnel, and in some cases the FAA; copies of developed material must be saved for potential historical review. Weather needs to be reviewed for flight origin, destination, and en route conditions. Alerts from the Weather Services at Suitland, Maryland and Kansas City, Kansas centers, from pilot reports, and station observations are sent to the origin station after review by the dispatcher. A flight plan is calculated using weather, origin/destination characteristics and conditions, aircraft particulars, flight speed and altitude, and fuel weight and amount. The basic flight plan must be filed with the FAA and a detailed plan must be sent to the origin station. Before electronic mail a standard flight plan was filed with the FAA and flight and station personnel. A standard profile was, by its very nature, general and conservative, leading to poor estimates of flight times and fuel burn. In-

creased optimization of the scheduling and use of aircraft, crews, and fuel has become possible by being able to send current information about weather and aircraft.

The management of aircraft movement is another area heavily serviced by electronic message transmission. Reported data are used for crew payroll, flight status, and accumulating the hours of aircraft flying. Traditionally, stations have used the teletype systems to inform downline stations and dispatchers of the out, off, on, and in times associated with aircraft movement. Summaries of these times by flight number were then sent by company mail to HDQ for analysis and storage. These same times are also recorded on Flight Logs which are kept on board the aircraft until a flight is completed. The Flight Logs are then placed in company pouches for delivery to HDQ. Flight Operating Systems have automated the retention and distribution of this essential information. U.S. Air has taken the additional step to incorporate this function into their integrated Crew Management data base system, taking advantage of current EDP technology to improve the access and management of the data. By real-time recording and automatic distribution, the urgency of the delivery of the Flight Logs has been reduced considerably and the station record delivery has been eliminated. A further enhancement has been the employment of on-board micro-processors (MPU) which have sensors allowing the collection of the needed out, off, on, and in times. Ground stations at the serviced airports are sent the accumulated messages over a VHF radio channel reserved for airline use. The ground receivers have a verify and retry capability allowing the checking and retransmission of messages, increasing the probability of accurate receipt of the aircraft movement information. The messages are then transmitted through the ARINC network to the industry Electronic Switching System in Elk Grove, Illinois and then on to the airline responsible for the original transmission. Airlines like Texas International and Piedmont use these messages in place of station generated messages. The benefits of the ARINC Addressing and Recording System (ACARS) are speed, accuracy, and improved employee productivity. Specifically, the ACARS system reduces the workload in the cockpit, at the stations, and in flight dispatch. The ACARS data can be input through communications interfaces into computers for payroll, aircraft flight time accumulation, and on-time performance reporting.

In the last decade several airlines have invested considerable resources in the development of integrated data base systems for the Maintenance and Engineering Divisions. Some of the more complete systems have respective costs as reflected at (a) Swissair—250 man years of effort, (b) Alitalia—140 man years, (c) Republic—100 man years, and (d) U.S. Air—80 man years. United is currently developing a system estimated to require more than 600 man years to complete. Republic Airlines and U.S. Air started work on their systems in the early 1970's. While numerous other airlines have on-line systems and are using data base concepts, the above two systems are further along in their development of fully integrated systems in the United States. These systems are similar to large manufacturing and inventory control

\*\* ARINC has the world's largest private line network servicing over 140 customers with more than 5 million circuit miles.

systems modified for the specific requirements of the airlines. Instead of multi-hour or even overnight delays in knowing the status of repairs and inventories, messages are sent to the schedule planners, shop foremen, inventory specialists and into the data base system allowing further reports and analyses to be accomplished. Automatic and demand messages are generated to order new parts when needed.

Through electronic message delivery more accurate and timely inventory control in the remote stations is achievable. Status of incoming aircraft relative to needed repairs, allowing improved scheduling of the work force and improved decision making to identify the best location for the repair can be accomplished. By replacing the manual process of scheduling aircraft repairs at each individual station with automatic preparation of schedules, work optimization and improved availability of the fleet can be achieved. Messages are sent to report available manpower and the estimated use of parts. Location of the aircraft and estimated time of arrival are obtained from the Flight Operating System. Reports are then transmitted to the affected stations showing the schedules and requesting movement and location of the necessary parts.

The Maintenance and Engineering Division is broken into several departments, each of which is a heavy user of electronic message transmission. One such department is Purchasing and Receiving which evaluates materials to be purchased, selects vendors, orders equipment and services, and takes receipt of purchased goods. Advancements in message delivery have resulted in lower inventory levels. Recently some of the larger suppliers of aircraft equipment have provided the ability for purchasers to access their data bases for stock levels, time to manufacture, and cost. Using terminals linked to their system, the entire cycle of phone calls and mail orders has been drastically reduced. Boeing, McDonnell Douglas, and Pratt and Whitney are three suppliers which offer such services to their customers. The overall benefits have been reduced manpower for handling orders, faster turn-around times, and fewer mistakes and misunderstandings.

By use of terminals attached to the integrated system, the processing and tracking of orders and receipts has resulted in the elimination of considerable amounts of paperwork and physical (vs. electronic) information delivery. When a shipment of goods is received, the receiving clerk updates the open order file for status, making available the pertinent information in the system. With this information the system can initiate payment, identify the storage locations for the material, and debit the budget of the requester. Also letters can be generated if the shipment is incorrect or not complete.

A message is transmitted to update the data base during the actual receipt process. Key information is then immediately available including the date of order, expected cost broken down by component for handling partial shipments and invoices, location material should be sent to, who placed the order, and budget assignment.

An important aspect of these integrated systems with access to message switching is the ability to use a given item of data to service multiple needs. This has the direct benefit

of consistency and manageability. A specific example of the above is the flight movement data from the ACARS system which has the following multiple uses:

- a) for flight dispatch—for the tracking and controlling of aircraft movement,
- b) for collecting time of use—for aircraft maintenance,
- c) for crew payroll,
- d) for on-time performance reporting,
- e) for aircraft performance analysis,
- f) for notification of down-line stations regarding flight progress.

The most recent innovations in message send/receive have increased the computer's role in the process and reduced the manual intervention of recording the times and typing out many of the messages. The direct benefit is reduced manpower, along with more accurate and timely receipt and availability of the data.

Engineering orders are directives to maintenance to make modifications to aircraft and include the detailed instructions to accomplish the modification. Detailed records are required relative to these Engineering Orders. In addition, the aircraft manuals must be updated to reflect the current status of each aircraft. Copies of the updated manuals are then printed and distributed to all affected groups including remote maintenance locations. For urgent transmission, facsimile equipment is employed. Improved techniques in digitizing and transmitting material directly from a page or picture has allowed more maintenance to be performed in remote locations and still fulfill the requirement of having a current copy of the reference manual on hand. Facsimile processing times have fallen over the last 10 years from around 30 minutes per page to less than 1 minute per page. Using a combination of facsimile and the integrated on-line system with text editing and storage, changes can be incorporated directly into the computer system, accessed by scheduling, and referenced by the maintenance foreman and mechanic. The paperwork and time delays for the old system have thus been effectively reduced to a minimum. Even the report of the Engineering Order accomplishment is directly input, thus being available for immediate review and analysis.

## RESERVATIONS

The capabilities of general airline reservations are relatively well known; thus, only a few examples are given to demonstrate some of the lesser known applications of reservations message transmission. Eastern Airlines developed a computer system to which most airlines subscribe, and submit descriptions of unclaimed bags, allowing the airlines to enter descriptions of the lost items and query the data base for likely matches. These queries are routed to Eastern by means of the ARINC network. If the item is located, the involved carrier is sent a confirmation message giving item location and contact point. Messages are then sent to make arrangements for delivery of the items to the desired location.

For years airlines have printed tickets in commercial accounts and travel agencies. These were generated by agents sending teletype messages with ticket information to the industry standard TTY Model 28 RO printer in the account location. In 1972, United developed the "Teleticketing"<sup>2</sup> capability to send tickets directly from the computer to a customer's printer at preselected times. Advances in automatic dialing and connect technology by terminal and communications equipment vendors allowed this cost effective service to be implemented. An aborted extension of this was developed by United and Braniff to send messages (tickets) to local Post Offices who would then mail the tickets to requesters. Technically the application was sound; however, objections by travel agencies and some control difficulties resulted in this application of electronic mail being shelved.

The biggest single expansion in the reservations services has been the development of travel agency services by several airlines. American and United currently dominate this offering which now includes the co-hosting of these services for other airlines on the American and United systems. Basically, the service includes the traditional reservations features; however, it now has been enhanced to handle itineraries, invoicing, and special reports. These systems provide a very cost effective shortcut to the traditional method of calling an airline's reservations office, handwriting tickets at the travel agency, performing credit checks, sending the credit card receipts to the credit card company for payment, and the eventual payment of the airlines in response to their invoicing. Today direct links between travel agencies, the airlines and the credit card companies are an early example of the merger of Electronics Funds Transfer with Electronic Mail.

## PROBLEMS

The airlines have been aggressive in the use of telecommunication networks for reservations, real-time processing of maintenance and engineering and flight operations messages, and for corporate message transmission. Millions of dollars worth of terminals and communications equipment are currently installed using the very efficient but limited ALC line discipline.<sup>†</sup> With the continued rapid introduction of new applications which go far beyond the original design specification of the networks and hardware, severe restrictions and costly redundancy are being experienced.

Computer and communications equipment reliability needs to be improved. Availability of 99.5 percent is still very difficult to achieve even at the central site. With the expanded networks automated detection, control and repair of the circuits is essential. A 95 to 98 percent availability using common carrier facilities is still a challenge, requiring sophisticated and expensive test equipment to assist the vendor in problem diagnosis.

Data security is becoming increasingly important. The preponderance of data being delivered across airline com-

munication networks is easily susceptible to compromise. As further use is made of these networks for vital company information and financial transactions, data will need to be protected. The use of the current line disciplines will severely hamper this protection. Accompanying security is the need for improved message assurance. Recent developments in computer vendor offerings will need to be employed to improve the consistent quality of the delivered product.

The airlines have been slow to upgrade to the new vendor network architectures like the IBM System Network Architecture,<sup>3</sup> DEC Network Architecture<sup>4</sup> or the Communications Industry X.25 recommendation.<sup>5</sup> The slowness can again be directly attributable to the high replacement costs and the lost efficiency for some of the systems with the highest performance requirements. Clearly, experience in past development will help for the future; however, due to the generally individual development of functions, and the solidly entrenched standards, integration and control of upgrading will be a sizable challenge.

## FUTURE

The airline industry was an early user of Electronic Mail. It has been building on its base of applications and experience. Looking at the new technology coming from the R&D labs of the computer and communications industries gives a view of future capabilities. Fiber Optics and Satellites combined with increased use of LSI and VLSI chip technology, along with advances in bubble memories, will make available larger data paths with increased intelligence. Public Networks offering packet switching services such as TELENET and TYMNET will gain users, saving the trouble and expense of dedicated networks. Common carrier offerings like DDS and ACS will make access and use of networks easier and cheaper. The airlines are just now moving up into the use of broadband channels such as their recent introduction of 56KB circuits. The airline industry requirement for real-time availability of information will force the continued expansion of the use of message switching.

This author believes the next phase of activity will be in further integration and interconnection of computer system data bases allowing more automatic generation of messages to service the operating divisions. With improved quality of digitizing techniques, facsimile processing for legal and administrative purposes will expand. Today's use of facsimile for sending contracts, personnel resumes, and technical manuals will also increase.

Unified networks are sure to be more heavily employed. The X.25 and SNA standards will be increasingly used. Applications will be modified to further reduce the manual intervention required today. Finally the increased logic capabilities of the computer and communications equipment are sure to be major contributors to the coming improvements.

Little has been said about the importance of the man-machine interface. The best offerings of new development can not be effective without taking into careful consideration the user. Today the preponderance of credit card, check-in,

<sup>†</sup> Limited to 64 characters due to the 6-bit structure of the code.

boarding, and departure information is input manually. The use of concepts like Point of Sale is a needed enhancement. The accumulation of station data is also a manual process. Distributed Data Processing systems will be further employed to minimize the effect of communications failures and increase the computing power available for airport automation.

## CONCLUSION

The airline industry is among the leaders in the use of electronic mail technology and concepts. The benefits have been rapidly realized, resulting in improved manpower efficiency, reduced aircraft inventories, improved availability of aircraft, and improved passenger handling.

The impediments to the employment of the rapidly improving technology are significant and costly. Timing will be dependent on when the return of investment for the new capabilities is sufficient to warrant the multimillion dollar

investment. The pace is apt to be slow relative to the addition of new applications; however, with further improvements in communication front end processors and distributed processing power, the ability to gradually upgrade the networks is possible and will undoubtedly take place.

## REFERENCES

1. Frost & Sullivan Limited, "The Electronic Mail Market in Europe," 106 Fulton Street, New York City, USA, Report #E312.
2. J. C. Goodlett, "The UPARS Network—An Extensive Network Servicing Several Diverse Applications," *Proceedings of The European Computing Conference*, London, September 1975.
3. J. H. McFadyen, "Systems Network Architecture: An Overview," *IBM Systems Journal* 15, No. 1, 4-23, 1976.
4. G. E. Conant and S. Wecker, "DNA, an Architecture for Heterogeneous Computer Networks," *Proceedings of the Third International Conference on Computer Communications*, Toronto, 618-625, 1976.
5. Public Data Networks, CCITT Sixth Plenary Assembly, Geneva, September 27-October 8, 1976, International Telecommunications Union, Geneva, Switzerland, Orange Book, VIII. 2 and supplements, 1977.





# Metamorphosis: facsimile communications, electronic mail and office productivity

by JOHN E. COCHRAN

*QWIP Systems,*  
Division of Exxon Enterprises Inc.

Information is a fundamental resource that can be leveraged to meet the demands of the imperative for productivity improvements in our offices.

Information is the primary element of ideas and creativity. New ideas, analyses, creativity, and synergy are the corner stones of improved planning and problem solving. . . . They are the foundations of consideration of more and unique alternatives, and they are the prerequisites for better decisions.

More accurate, more complete, more comprehensive information, developed and delivered more expeditiously and more efficiently to the real users of it, is the highest pay-off ingredient for better management.

Better management . . . of time, people, money, facilities, tools, energy . . . is our hope for improving productivity in our offices during the eighties.

The office worker has finally gained the majority of the total United States labor force. Secretaries, executives, professionals, managers, administrators, all create, use and communicate information. These workers make more than 100 billion telephone calls, and they produce 72 billion documents, while maintaining another 300 billion, each year. And, these volumes are growing at a rate of 20 percent annually. The problem of productivity in our offices is not caused by insufficient production of information. Indeed, you and I are usually overwhelmed by the volume of it.

The forty-five million or so American workers who use information are constrained not by the amount of it, but either by the timely availability of it, the facility for delivering it to the right users of it, and by the prioritization of it so that it can be useful within the limits of our human abilities to manage it as a valuable resource.

How frequently have you and I said the following. . . .

“If I had only known those numbers when I made that decision”

or

“Why didn’t you bring those facts to my attention when I needed them”

and

“Gosh, I remember now that you told me that, but I had other things on my mind and I just didn’t remember what you’d said.”. . .

The information explosion in our country has become so much a part of our lives that managers now spend 95 percent, and professionals 63 percent, of their time communicating it in written and oral form.<sup>1</sup>

But what are we doing to support these people who represent one of the greatest opportunities to improve the vital productivity problem in our country?

Productivity in the office has risen just four percent over the past decade, compared to a 90 percent improvement in industrial productivity. Increases in farm productivity are legend.

However, the average American farmer is supported by \$54,000 worth of capital equipment and the average factory worker by \$31,000. On the other hand, the office worker . . . remember, now in the majority of our work force . . . is supported by only \$2,300 worth of capital equipment.<sup>2</sup>

I’m sure some of you have heard these kinds of numbers before, and since it’s not new news, you may be saying, “So what?”

My point is simple;

Our country has an economic problem called productivity. It can be measured in our expense budgets and tracked by the escalating inflation rate. Our big target for improving productivity is the office. Once inside the office, our bullseye is the manager and the professional . . . not the secretary or the clerk. Our real target is the decision maker, the influencer and the creator. Our goal must be to supply these people with their most important tools. Certainly, their single most important tool is information. They cannot manage without it.

As the facsimile industry leader and spokesperson, we at QWIP Systems believe facsimile communications offers a significant solution to the productivity problem. We believe that facsimile communications is the most efficient way to move information to its real point of need so that it can be utilized by people to make more informed, more timely, hence, better, decisions. In this country alone, almost a quarter million workers have discovered facsimile to be a prerequisite of the management process.<sup>3</sup> And they represent only the tip of the iceberg.

Facsimile communications has been around for over 135 years.<sup>4</sup> However, the real market, and the one with the greatest promise for improving information utilization, did not begin to develop until the 1960's.

The first users of facsimile products during the modern era generally had a highly applications oriented need for transmitting graphic information like charts, photographs, advertising copy and the like. It was obvious that since facsimile re-created the image of an original document, it was inherently the most accurate way to communicate information. Also, since it produced a hard copy reproduction of the original document, the information that was communicated had lasting value . . . it could be filed, copied, manipulated, recalled and annotated.

This inherent value was perceived by entrepreneurs and major businesses alike, and in the late 1960's firms such as Xerox, and new companies, like Graphic Sciences, entered the market.

By 1974, several firms were in the market, including 3M, with the Japanese giant Matsushita for a partner, and a new venture of Exxon Enterprises Inc., called QWIP Systems. A new term had emerged as a descriptor of a market subset, termed convenience facsimile.<sup>5</sup>

Indeed, by the mid-70's, it was obvious that facsimile communications had gained industry stature, and that customer usage, while still applications driven, was becoming less specialized. Facsimile products were being used to communicate textual information as well as graphical, and had the capacity for doing both on the same document. No other communications medium could cost effectively do this for convenience use. Facsimile products were being moved out of the laboratory and specialty areas and mailrooms, and into the offices. Thousands of users began locating the equipment on desks and credenzas when they could obtain products that were attractive enough to fit into their "human space."

By the late 1970's, three distinct market segments had established themselves within the facsimile industry. QWIP Systems had taken possession of the low end of the market with its 4,6 minute highly portable products, while racing with Xerox, Graphic Sciences, now a division of Burroughs, and 3M in the medium speed, 2,3 minute market segment.

In addition, several companies had created a third, high speed, market segment with the launch of several new products capable of transmitting documents in a minute or less.

Today, buyers can purchase the low speed machines for \$500 to \$2,000, the medium speed machines range in price from just under \$2,000 to \$8,000. And the new high speed, one minute and subminute machines are the very expensive darlings of our industry, cost as much as \$15,000.

It is our belief, at QWIP Systems, that the low speed market has matured, with a domestic population of just over 160,000 units installed. The medium speed segment of our market appears to be the one with the greatest growth potential during the next five to ten years. Vendor congestion and competition in the high speed segment, the high price of the units, and the significant problem of general lack of communications compatibility between different brands, appear to be limiting factors to us in this segment during the next few years.

In order for facsimile products to meet the information

needs of today's business managers, they must communicate. Different vendors must adhere to the international facsimile standards established by the International Telegraph and Telephone Consultative Committee (CCITT), such as those published for the Group II machines, which we at QWIP Systems have adopted for our medium speed products. The adoption of these standards in product design is a fundamental responsibility that is shared by all manufacturers of facsimile products in order to satisfy the needs, in fact, the demands, of our customers. In our opinion, the utilization of dissimilar protocols in some facsimile products has limited the natural, even more rapid expansion of our market, that could have otherwise been experienced.

Our market projections in the various segments I have described lead us to expect that the installed base of the low speed units will grow from 160,000 at the beginning of this year, to about 175,000 at the beginning of 1985. The medium speed unit base will grow from about 25,000 units to well over 300,000 during the same time period, and we expect the high speed machines to grow from just under 10,000 units to about 30,000.\* We at QWIP Systems also feel, however, that these are very conservative projections.

Indeed, the advent of new lower cost technology, new intelligent communications networks, and more human engineered facsimile products, coupled with government and industry's dramatic requirement for white collar productivity improvement, could literally explode these projections.

Our industry's challenge, then, will continue to be the effective management of our growth, the delivery of reliable products . . . and the provision of excellent service to our customers.

Before I leave this area of the market, I do want to briefly mention two very important facsimile aftermarkets that are occasionally overlooked. In our business, the common carriers and the specialized common carriers find facsimile network services to be lucrative. Our rule of thumb is to project that each installed facsimile machine will generate its value in communications services revenue each year. The aftermarket for consumable materials, supplies and service is the second most significant area for revenue, mostly accruing to the facsimile manufacturers. In fact, we estimate that by 1985, industry revenues from materials, supplies and service will approximate twenty percent of the annual machine revenues from outright sales and rentals.\*\*

Now, let's spend the rest of our time together discussing the probable evolution of our industry, and its impact on electronic mail.

Facsimile in the next four to five years will take advantage of new technology as well as enhancements that are developed in the convenience facsimile machines similar to those in existence today. The convenience facsimile market will continue its rapid growth and, in addition, there will be an emergence of multi-functional products which will merge facsimile with other automated office functions.

Let us now examine the basic technologies involved in facsimile and some of the general trends occurring in these

\* Estimates based on Market Projections, QWIP Systems, Division Exxon Enterprises Inc.

\*\* Estimates based upon projections of QWIP Systems, Division of Exxon Enterprises Inc.

areas:

*Scanning* is the process of analyzing the copy into a serial stream of light intensity variations, which in turn are converted into voltage variations.

Traditionally, the process has used electromechanical and electrooptical devices; however, the trend is toward solid state, laser and fiber optics. Recent developments such as photodiode arrays and charge coupled devices (CCD's) will pave the way for high speed scanning, improved reliability and lower production costs while being compact, stable and quick to respond.<sup>6</sup>

*Printing* is the process in which the transmitted image is reconstructed by converting the receive voltage variations into marks on the paper. Some of the more recent developments in non-impact printing include laser printing, precision matrix printing, ink jet, and improved thermal printing, which will enable high quality images on bond-like paper. These processes offer speed capabilities that range from 10 characters per minute to 45,000 lines per minute.<sup>7</sup> Advanced machines will be capable of making multiple copies upon receiving remote commands. Many transmitted documents are business forms which consist of fixed formats and variable information. Future machines will make use of an electronic overlay technique which will enable users to transmit the variable information only, thus reducing communications time and cost.<sup>7</sup>

*Storage:* The storage requirement for a typical page with a resolution of 96×96 lines per inch requires a capacity of one million bits. Data compression can reduce this requirement to approximately 140,000 bits. Typical storage costs have ranged from approximately 0.001 cent per bit to 0.7 cent per bit, depending upon technologies employed, and are expected to drop. This trend in cost reduction will enable practical store and forward systems. New solid-state memories, such as magnetic bubble memories, are beginning to become attractive in some office product applications as a result of their ability to store large amounts of information in a non-volatile mode.

Optical/Laser memories are capable of very high storage densities but are still only capable of intermediate speeds and they are somewhat limited to permanent archival-type storage applications.

*Electronic Controls:* The constant effort of electronic component manufacturers toward the design of standardized off-the-shelf building blocks such as LSI chips, memories, microprocessors, interface chips, peripheral chips, etc., has permitted lower product design and development costs.

Electron beam or x-ray lithography will improve the manufacturing process, thus permitting a further increase in gate density per chip. This will result in more powerful localized processors, thus permitting more functions at the terminal at lower costs.<sup>8</sup>

*Communications:* The public switched network has been the prime communication link for facsimile and electronic mail. Analog transmission in two minutes is now commonplace. Improvements in modem technology have enabled digital transmission at 4,800 bits per second. Up to 9,600 bits per second on the switched network is now possible; however, the cost of these modems has been substantial. LSI technology as well as a recently announced analog micro-

processor is expected to drastically reduce the cost of signal processing and the equalization functions of these modems.<sup>9</sup> The cost of digital facsimile products will drop, making digital devices practical at the point of need.

In addition, specialized common carriers and value added networks such as ITT's FAXPAK, Xerox' X-Ten and SBS, utilizing different technologies such as satellite communications and packet switching, will make higher speeds available at competitive costs. The advent of CCITT standards, such as the recently approved Group III facsimile standard in Kyoto, Japan, will insure compatibility on an international basis among various vendors' equipment.

The above technological trends coupled with a market demand for more office automation will result in the existence of two families of products; the convenience facsimile terminal and the multi-functional office terminal. The convenience fax terminal will continue to offer more and improved features and yet will remain within a price range which permits cost effective applications.

According to certain market analysts, in addition to the convenience facsimile device, the multi-functional terminal, which is compatible with existing machines, will emerge. This terminal will be capable of resolution in excess of 300 lines per inch, producing correspondence quality copy. The unit will also interface with other communicating office products such as word processors, TWX/Telex, and electronic data processing devices; thus making this a shared scanner-printer with both text and graphic capability. The terminal will provide local photocopying and will have storage and retrieval capabilities, as well.<sup>10</sup> Some of today's products such as the IBM 6670, Xerox 9700, Wang IP41L and Toshiba L2017 already have the capability to perform several of these functions. For example, the IBM 6670 utilizes a laser printer, has multiple copy capability, can transmit and receive over ordinary phone lines, and will interface to the IBM 370 EDP system and the IBM Mag Card II typewriter. Although the 6670 does not currently perform graphics nor can it accept facsimile input,<sup>11</sup> it is apparent due to its laser scan type printing that graphic capability could be easily added.

When evaluating products, this new breed of facsimile device (multi-functional terminal) could become a key component in the electronic office. Facsimile is the only product having the ability to communicate both graphics and text, making possible the transmission and hard copy generation of an infinite variety of data. This data can originate from original documents, EDP systems, word processors or other facsimile terminals. The device will minimize communication cost by permitting after hours transmission of data through the use of an autodialer and a store and forward option, as well as allowing for the utilization of value added networks which also permit communication with non-compatible equipment. In addition, the unit will provide convenience copying from a hard copy original.

The system, of course, will include the use of remote low cost portable facsimile terminals. This integrated system should begin to find market acceptance later during the 80's.

I've already characterized our industry's history and provided an overview of what is to come.

We have seen the change, and expansion, from highly

specialized products and customer applications to this new era of so-called convenience facsimile. We've watched the transition of products from low speed to faster speeds, and we've touched on the customer demand, and subsequent supply, of products that are designed to provide more communications compatibility with dissimilar brands.

I believe the next few years will be no different from what has now become commonplace . . . more changes . . . at a faster rate. We call it industry metamorphosis.

To illustrate my point, let's evaluate buyer motives that should find expression in new products and services during the next few years. The first, and most obvious, that I've already introduced to you, is the imperative for improved office productivity.

New facsimile products must then be provided that offer convenience, ease of use, automated paper handling, micro-processor machine control, and automated telephone management and utilization. These new products must be designed to solve the real productivity problem . . . they must target the information they communicate to the real point of need—the manager and the professional. These new products must be so cleverly designed that they fit within the price parameters of the purchasing decision makers, so they can be demonstrated to be cost justifiable for the white collar worker's personal use. And they must be attractive enough to fit into the personal work space of the people who use them.

Facsimile products, then, will be friendly and easy to use. They will be transparent to the user and they will be cost justifiable to the buyer. In many cases, they will expedite information delivery to the *real* user, rather than to the mail-room attendant or the clerk who is often positioned between the communicators . . . who, because of their positioning, tend to delay or reprioritize the delivery of time critical messages.

The next few years of our business will be characterized by more general customer use. Our studies at QWIP Systems have indicated that in the past about 90 percent of all facsimile use was applications specific and a like percentage of use was for intra-company communication. Our customers are now beginning to tell us that they need to communicate with *their* customers, *their* business associates and *their* suppliers. And they want products that can be used to send letters, order entry documents and graphs, without as much emphasis on special uses. Communications compatibility with a wide variety of competitive products is now an essential rather than an added feature.

Now, what role does facsimile play in the emerging electronic mail—electronic office—marketplace? I feel we need to revisit the fundamental reason for the emergence of this new market, and the basis of this report. . . . The prerequisite for white collar productivity improvement . . . to complete our perspective and to answer the question. Heretofore, the most attention for instituting office automation to improve productivity has been in the product areas of "word processing" and "machine dictation." Studies show that typing productivity can be improved by orders of magnitude from 200 to 500 percent by word processing systems, and author composition can be improved by 30 to 50 percent

through the use of machine dictation, although user resistance has retarded the exploitation of dictation as a real productivity improver.

When evaluating the overall distribution of labor costs in American businesses, the secretary-typist represents only about 6 percent of the total. On the other hand, non-clerical labor costs are about 66 percent of the total.

I am not belittling the development of word processing products as productivity problem solvers—their outstanding results in the typing area speak for themselves. The real opportunity for leverage, though, according to the noted analyst, James Bair,<sup>1</sup> and we agree, is in the support of the manager and professional. To illustrate, United States business costs for secretary-typist labor exceeds \$4.4 billion, contrasted to non-clerical labor costs of about \$250 billion. The leverage then can be demonstrated on the basis of labor costs alone.

As I stated in my opening remarks, most of what managers and professionals do each day is communicate information. Bair has estimated that electronic mail has the potential of saving managers and professionals about 2 hours a day by improving the way they communicate, by making communications more efficient, and he has projected that the resulting increase in productivity could save U.S. business up to \$62 billion a year. His treatment, while academic and perhaps optimistic, is nevertheless interesting . . . and appears to be valid.

We believe that facsimile products, as ubiquitous electronic mail communicators, offer outstanding benefits to the electronic mail customer. First, original documents, whether graphic or text based, can be transmitted. Second, once transmitted their messages have lasting value and can be copied, filed, annotated and redistributed with ease. Next, original documents can be communicated without the need for expensive reformatting. And, last, facsimile products can be inexpensive. Ours are, and they fit into the work station without intrusion.

Yes, we are all a part of the metamorphosis of our industries, the office environment and the evolution toward a more productive place in which to work. Those who succeed in this changing market will demonstrate flexibility, awareness of the importance of communications and will recognize the real targets for their products. They will have developed products that are responsive to market needs, yet they will boldly drive their markets with a keen sense of leadership. They will have assembled marketing and engineering organizations that are capable of customizing systems and networks of their products, even including those of their competitors, if necessary, to meet customer demands.

They will have made the connection between information and those who must have it, use it, and make decisions based on it.

## REFERENCES

1. James H. Bair, "Communication in the Office of the Future: Where the Real Pay Off May Be," Submitted to the International Computer Communications Conference, Kyoto, Japan, August 1978.

- 
2. Morris Edwards, "Automated Office Adds Muscle to White Collar Productivity Drive," *Communications News*, May 1979.
  3. QWIP Systems, Division of Exxon Enterprises Inc., Proprietary Data.
  4. Daniel M. Costigan, *Fax, The Principles and Practice of Facsimile Communication*, Chilton Book Co., 1971.
  5. *Facsimile markets*, International Resource Development, Inc., April 1979.
  6. IDC Report, "Fax Marketing Forecast," March 1979.
  7. Elizabeth A. Hughes, "A Printer Primer," *Computing, Inc.*, Fall 1979.
  8. Rajchman, J., *Science* 195, 1223 (1977).
  9. Robert H. Cushman, "The Promise of Analog Microprocessors; Low Cost Digital Signal Handling," *EDN*, January 5, 1980.
  10. The Yankee Group, The Second Annual Symposium on Implementing Advanced Office Information Systems, June 10, 1979.
  11. Robert Conrad, "Intelligent Copiers and Image Printers for the Office of the Future," Strategic Business Services, 1979.
  12. Bolt, Beranek & Newman Inc., Communication Technology Forecast—Report 4037, January 1979.



# Texas Instruments computer communication network and its support for the automated office

by JOHN W. WHITE

*Texas Instruments Incorporated*  
Dallas, Texas

This paper will give an overview of the Texas Instruments Incorporated worldwide computer/communications network (as of November 1979) and specifically emphasize its support for electronic mailing, electronic filing, and network connected word processing capability.

## TEXAS INSTRUMENTS THE COMPANY

Texas Instruments is a multi-national corporation with forty-eight major plant sites in eighteen countries. The map of the world shown in Figure 1 depicts the major manufacturing sites and the year of initiation of operation at each of these locations.

Texas Instruments is a diversified company with the Semiconductor Group offering micro-electronic systems, integrated circuits, micro-processors, and memory systems; the Digital Systems Group offering mini-computers, terminals, and distributed processing systems; Consumer Group offering digital watches, calculators, and electronic learning aids; Metallurgical and Electrical Products Group offering precision metals and electrical control products; Equipment Group manufacturing defense systems such as missiles, laser guided bombs, and high precision radar systems; Geophysical Exploration Group providing petroleum exploration services, both land based and marine; and the TI Supply Company, a distribution arm for the products manufactured by the other divisions of the company.

Texas Instruments is a growth company as shown in Figure 2. In 1946 when the laboratory and manufacturing division of TI's founding company, Geophysical Service Incorporated, was established, net sales billed were \$3 million. In 1978 net sales billed were over \$2.5 billion. Some of the TI's major products which have helped TI's growth were the introduction of the first commercial silicon transistor in 1954, the single chip micro-computer or calculator on a chip in 1971, the bubble memory device in 1977, and the 64K dynamic random access memory in 1978. One of TI's newest innovations, announced just this year, is the 64K E-PROM which should prove to be another major product to add to this list.

## INFORMATION SYSTEMS AND SERVICES

The Information Systems and Services organization of TI is chartered to provide computer systems and communications capability to TI worldwide. The Information Services organization operates the Corporate Information Center in Dallas, operates the Voice and Data Communications Network and is responsible for the advancement of this computer/communications network.

The Information Systems organization has responsibility for working with user groups to define, develop, and maintain applications systems supporting engineering, manufacturing, marketing, and the corporate control functions.

To get a better perspective of this Information Systems and Services organization I would like to describe briefly the computer/communications network that offers these services. First we will look at the central computer center that supports this organization, the Corporate Information Center (CIC).

### *Corporate information center*

The Corporate Information Center is located in Dallas, Texas, and it is driven by six IBM 3033 mainframes. These computers have access to over three hundred twenty disk drives and eighty tape drives. They operate in a loosely coupled multiprocessor environment under JES3. In order to provide the high degree of reliability that is necessary to support the corporation, software and hardware is fully tested on an IBM 3031 in a VM environment prior to being placed into production.

As shown in Figure 3, two hundred seventy-five remote job entry terminals are interfaced to CIC and have access to batch processing capability on all six of these large mainframes via the JES3 job entry system. There are over eight thousand inquiry terminals that have access to the inquiry terminal network processing under IMS on two of these CPU's. These two IMS CPU's interconnect via channel-to-channel adapters through the Multiple Systems Coupling feature of the Information Management System. Many of these terminals also have access to the TSO time sharing system which also runs on two separate processors in this complex.



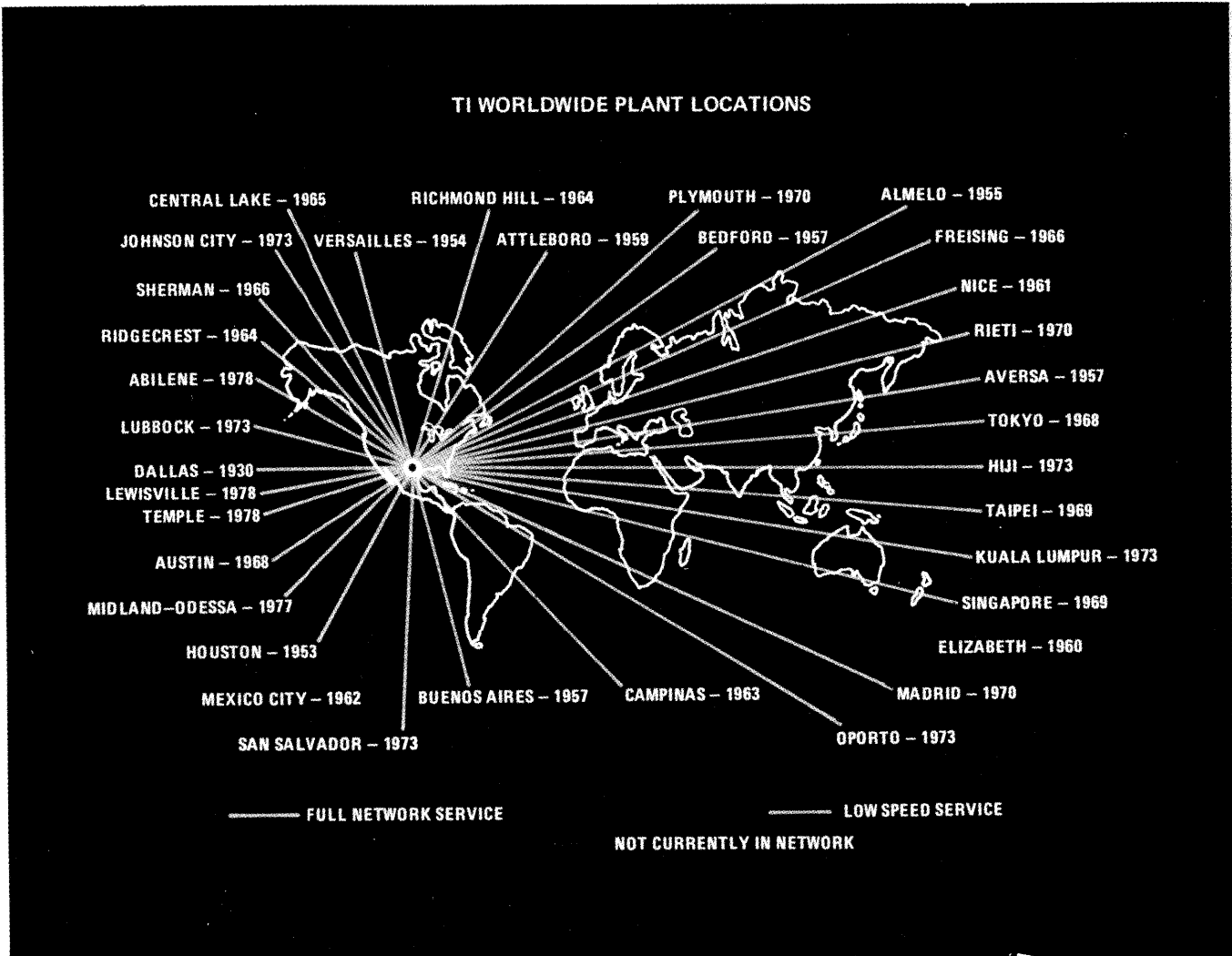


Figure 1

### Data communications network

Two hundred seventy-five remote job entry terminals, 8000 inquiry terminals, and 135 distributed processors are connected to the Corporate Information Center and to each other through a worldwide data communication network shown in Figure 4. The international network is made up of high speed trunk lines managed by communications processors with service emanating from our computer center in Dallas. A 50 kilobit line between Dallas and Singapore serves the plant sites in Singapore, Malaysia, Taiwan, Australia, and the Philippines. A 19.2 kilobit connection between Dallas and Tokyo serves the TI plant sites in Japan. To Europe, we have three 50 kilobit lines plus multiple 9600 baud lines which are used for voice communication and as emergency backup for the 50 kilobit links. Central America and South America are served with 9600 baud circuits to El Salvador

and Panama with low speed lines (teletype) to Argentina and Brazil.

The TI International Network is managed by a packet switching system based on TI's 980 computer. The processors that manage this network are referred to as TICOG (*TI Communications Grid*) processors shown schematically in Figure 5. This network allows the multiplexing of multiple devices and functions over a single communication line by supporting remote job entry terminals and distributed processors as well as direct interface for inquiry terminals. Alternate routing provides improved reliability of the communications network. Failure of a link within the network is detected within twenty seconds and traffic on that link is routed through the appropriate alternate route. By utilizing a continuous-transmit/selective-retransmit protocol we get a four to one improvement in line utilization over conventional protocols on satellite links. Priority queueing for the

interactive functions (IMS and TSO) minimizes the impact of communication delays within the network.

*Distributed computing*

In order to provide the improved reliability, improved responsiveness, and the lower cost of specific processing functions, this network is supported by 135 full function distributed processors supporting applications such as order entry, purchasing, receiving, material control, work in process, automated warehousing, and material accountability. These distributed processors primarily provide real time interactive interfaces for high performance functions.

The current network supports basically a three level computer hierarchy with the CIC representing Level I major mainframe capability, and distributed processors at Levels III and IV. We are planning for a computer network as shown

in Figure 6 giving us a four level computer hierarchy with IBM mainframes at Levels I and II and TI 990 mini-computers at Levels II, III, and IV within that hierarchy.

Level I systems will provide corporate consolidation and globally accessible data bases. Level II will be site processors supporting an individual site and possibly very near access areas such as field sales offices. Level III supports small sites and also functional areas such as a manufacturing line or an inventory control point within a large manufacturing operation. These Level III systems will also be used as the primary processor in field sales offices for order entry, finished goods, inventory control and financial systems support. Level IV systems will be used as interactive work stations for engineering design functions and as word processor work stations which will be discussed later in this paper.

The current TI network supports over 135 distributed processors that are connected to the TI master network. This collection of distributed processors is growing rapidly at the

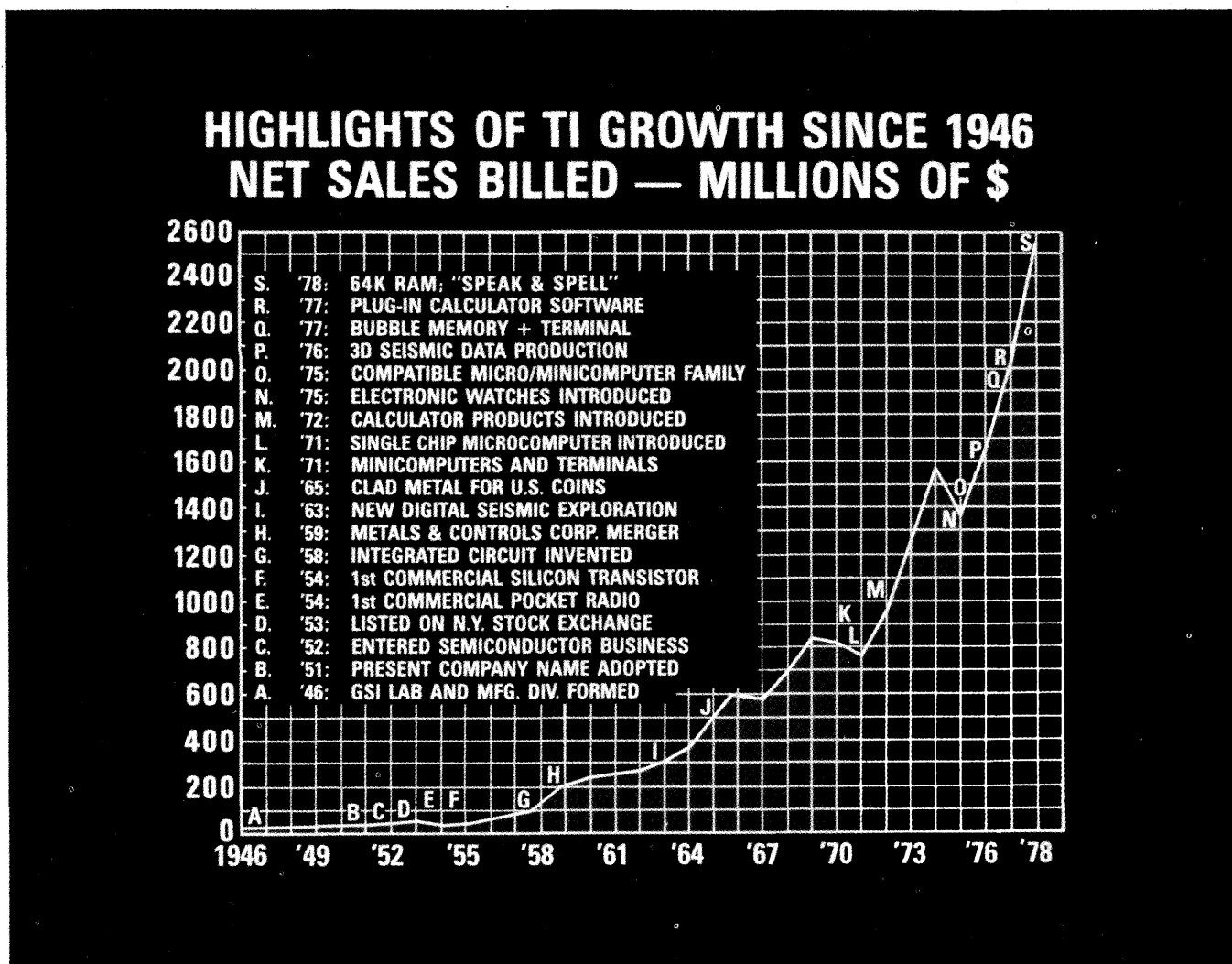


Figure 2

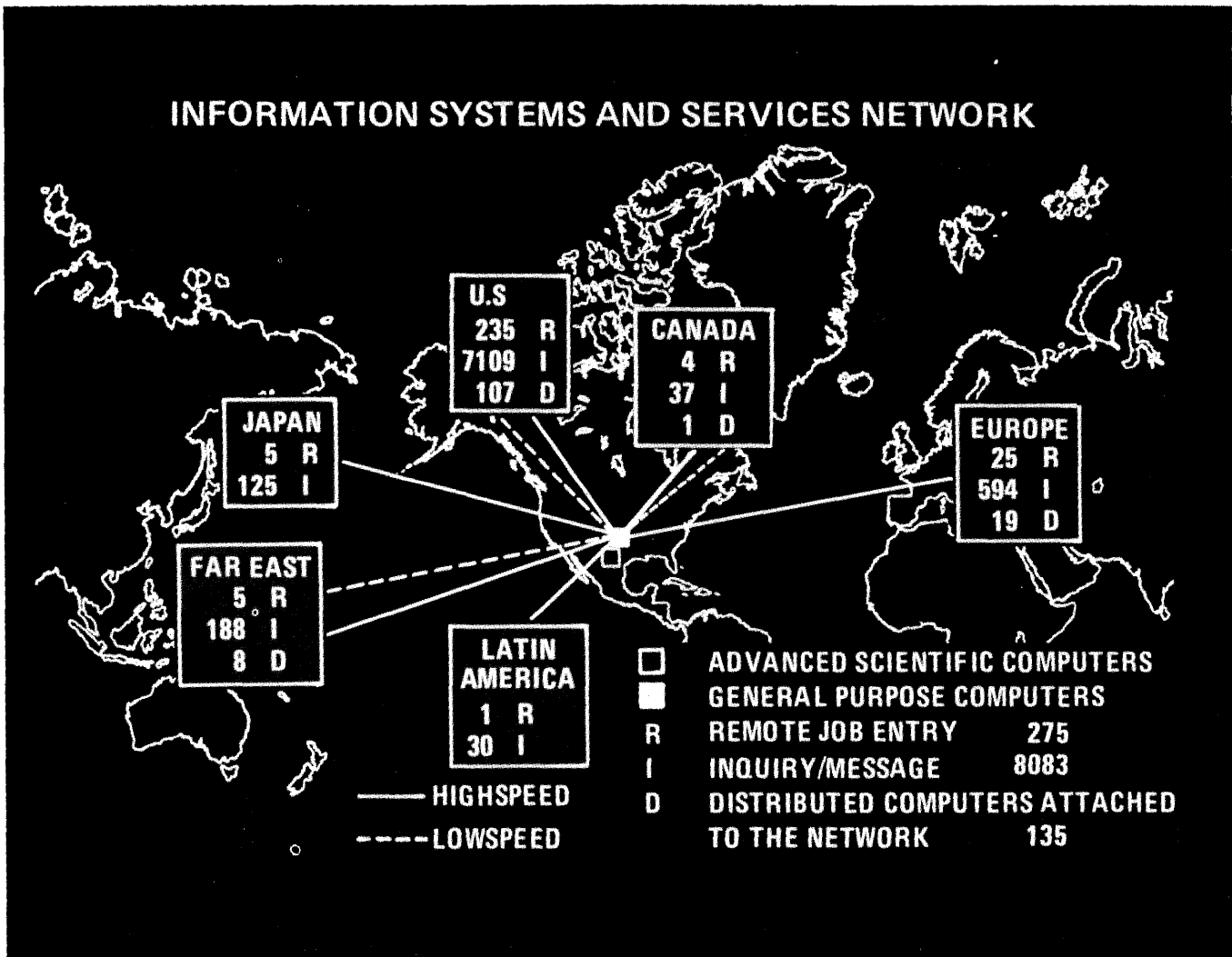


Figure 3

present time and is expected to accelerate in growth as more comprehensive functionality is developed for these distributed processors.

#### THE AUTOMATED OFFICE

This section will discuss how this network provides support for the automated office functions of word processing, electronic mailing, and electronic filing.

##### *Electronic mailing*

The electronic mailing function at TI is supported by a system referred to as MSG which has been in service since 1971. MSG is used for document preparation, storage, dis-

tribution, and redistribution of message text. It supports a broadcast capability allowing a single message to be sent to multiple locations. It offers security for storage, access, and distribution and supports a direct interface to the public telex network. One feature of the MSG system which is now proving quite effective is the ability to insert messages into the MSG distribution system from other application systems. This is possible since the MSG system runs under our full function IMS system and therefore has realtime access to all inquiry terminals within the TI network.

These application systems insert messages into the MSG system to indicate such events as the completion of batch processing cycles and the subsequent availability of reports from these management systems. MSG also supports the capability for individual electronic post office boxes so that one can manage personal messages and get access to these personal messages from any point within the network. Mes-

sages within the MSG system are retained in an on-line queue for approximately ten days before being purged.

Figure 7 shows the number of MSG transactions per day increasing from ten thousand per day in year-end 1977 to an average of twenty thousand a day in August 1979. The cost per copy has decreased throughout this time period due to improved cost effectiveness of computer and communications capability offering the service and economy of scale benefits that accrue from increased shared usage.

The MSG system is used by a wide variety of TI personnel including engineers, secretaries, planners, managers, controllers, and even ships at sea. Our geophysical exploration business has a fleet of marine exploration vessels which communicate via Marisat to the CIC and therefore have access to the MSG system as well as other management systems within the Corporate Information Center.

*Electronic filing*

The primary system that is used with TI for electronic filing is referred to as TIOLR (*TI On-Line Reporting*).

TIOLR was originally conceived solely as a cost effective alternative to printing. This system was designed as a hierarchical reporting structure accessible on-line by any terminal in the network and therefore offered additional capabilities. Some of the current applications are computer generated reports, electronic newspaper, reference information, systems documentation, and it is also used as an input mechanism for data collection functions.

The basic element of the on-line reporting system is the page with size set at a maximum of 66 lines by 132 characters. Figure 8 portrays the data structure which allows for a large number of reports, forty generations of each report, and 5600



Figure 4

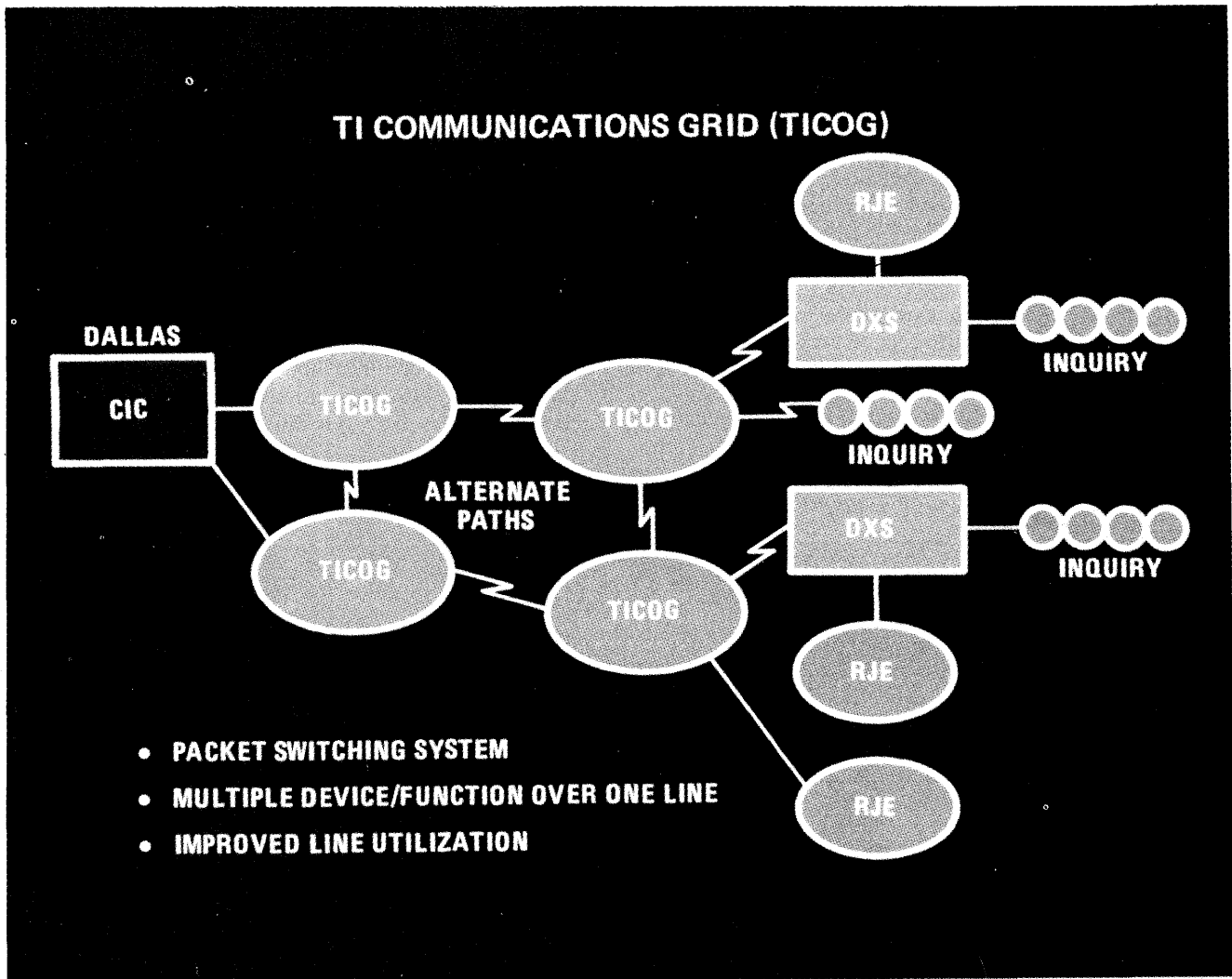


Figure 5

chapters per generation, sections per chapter and pages per section.

TIOLR is used for electronic filing and on-line reference allowing users to view one page at a time on a video screen with individual page addressability. Chapter, section, and page flipping with large page scrolling on a small screen is supported by function keys. Reports can be created, updated, and displayed on-line. Batch capability can be used for creation, update, and volume printing from this on-line file. Information security is offered at the report, generation, and page levels for separate read and update authority.

The TI On-Line Reporting capability was introduced in the third quarter of 1977 and its volume has increased dramatically especially during 1979 as its use became prevalent throughout TI. Figure 9 shows that the number of reports that are currently stored on-line is somewhat over 6000 and the amount of data stored is approximately four million

bytes. Cost per transaction has reduced from 3.9¢ in 1977 to 3¢ by year-end 1979.

#### *Word processing*

Word processing at TI is served through a TI internally manufactured word processor that is used for document preparation and filing. This system is connected to the TI Communication Network either by acoustic coupled dial-up or by a direct dedicated connection.

The TI word processor is a TI 990 based mini-computer with full page CRT, and letter quality printer as depicted in Figure 10. The word processor supports two work stations, each with a dedicated user floppy disk. The system may function in a standalone mode or connect via the computer/communications link to other word processors, distributed

processors, or to the Corporate Information Center. The communications interface utilizes a floppy disk to stage information to and from the network. This allows communications to operate in a background mode, freeing the word processor work station for foreground text preparation in parallel with communications.

When operating in the standalone mode the user can perform text editing, local document storage and retrieval from the floppy disk and printing from the floppy disk. By connecting the word processor to the network, additional functions are provided including: interactive terminal emulation for access to either IMS or TSO, and background communications for direct electronic mailing and remote electronic filing via the MSG and TIOLR systems.

There are currently 202 word processors in TI's network.

### IMPLEMENTATION FACTORS

TI has established corporate growth goals of 3 billion dollars in annual sales in the late 1970's and 10 billion dollars in annual sales in the late 1980's, as shown in Figure 11.

This growth thrust has stimulated the development of the TI worldwide communications network and its hierarchy of logic and memory. The communications network is an essential ingredient for implementation of affordable multinational integrated management systems, including those used for electronic mailing and filing.

As a result of this thrust, the TI Distributed Processing Network has grown significantly since year-end 1977 when we had 39 distributed processors connected to the network. Figure 12 shows this growth. By year-end 1978 we had 70

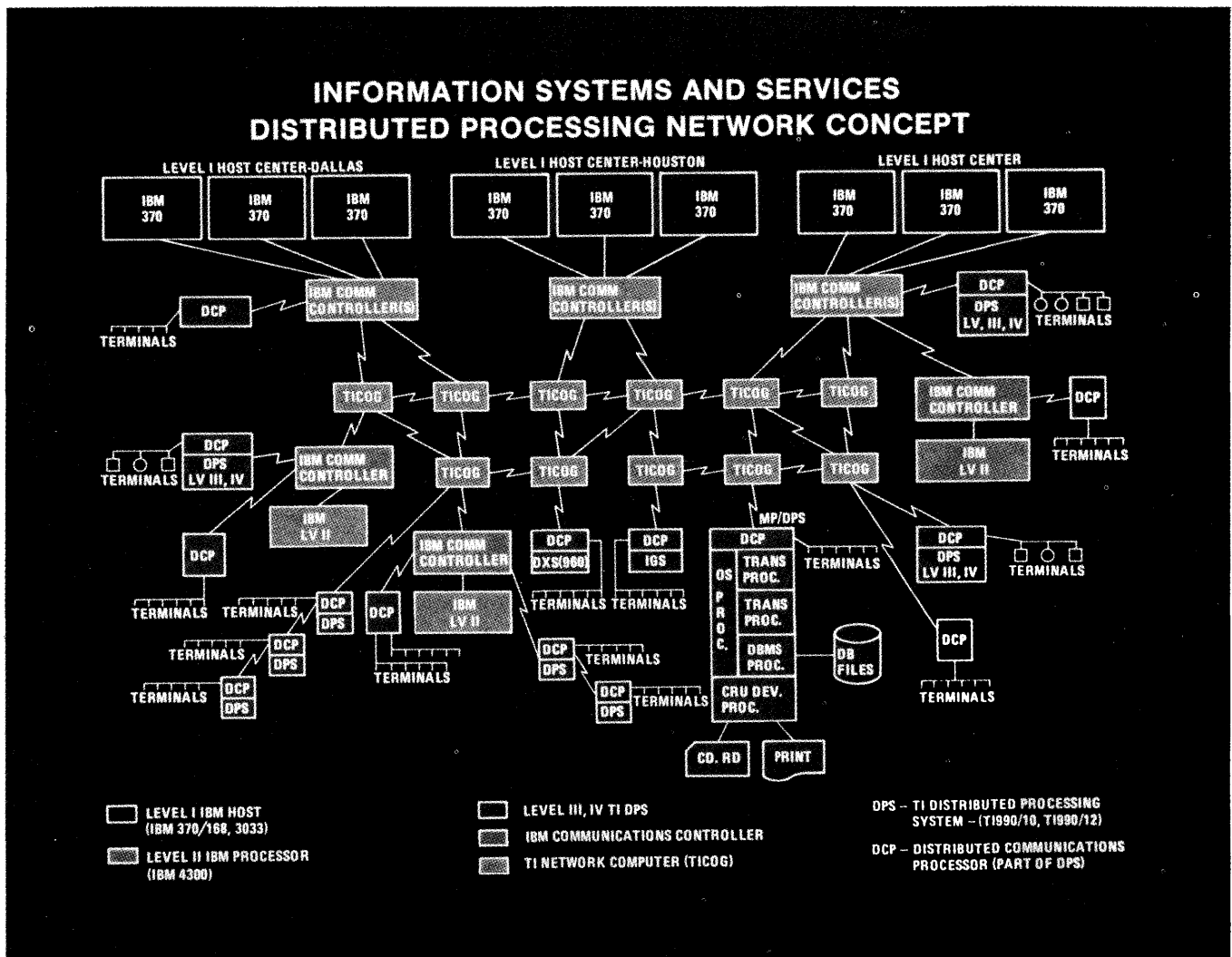


Figure 6

**TEXAS INSTRUMENTS  
INCORPORATED**

**TI USE OF ELECTRONIC MAILING AND FILING  
TIOLR VOLUME AND COST**

	<u>YEAR END 1977</u>	<u>YEAR END 1978</u>	<u>AUGUST 1979</u>
<b>NUMBER OF TIOLR TRANSACTIONS PER DAY</b>	<b>5,000</b>	<b>25,000</b>	<b>90,000</b>
<b>NUMBER OF TIOLR REPORTS STORED</b>	<b>350</b>	<b>3000</b>	<b>6200</b>
<b>AMOUNT OF TIOLR DATA STORAGE (MEGABYTES)</b>	<b>175</b>	<b>2500</b>	<b>4300</b>
<b>COST PER TIOLR TRANSACTION (1000 CHARACTERS)</b>	<b>\$ .039</b>	<b>\$ .031</b>	<b>\$ .030</b>

Figure 7

distributed processors connected to the network and as of November 1979 there were 135 network connected distributed computers.

In 1978 we saw a significant growth in our terminal population, more than doubling our total installed base. The percentage growth rate of terminals has decelerated in 1979, however we will install more terminals in 1979 than were installed in 1978. Network accessibility by a relatively dense terminal population is a key element of success of electronic mailing and filing.

In summary, the technology needs for the success of electronic mailing and filing are: 1) an inexpensive communications network (packet switching internationally in TI's case) which is accessible to a large number of users, 2) multi-system access from each terminal in the network, and, 3)

word processors which are compatible with general purpose terminals. At TI we have developed each of these.

#### THE BENEFITS

At TI we have seen a number of benefits from using the three key office automation systems.

The MSG electronic mailing system is used by the secretaries to transmit messages while continuing to perform text editing in the foreground. This reduces copying time, messengering time, and improves the speed of communication.

Benefits to the manager and professional from MSG elec-

tronic mailing include the speed of communications and the privacy that the electronic mailbox offers. One of the key contributions is the synchronization of communications worldwide. The system allows one to place a written copy of a document into a manager's in-basket wherever he is located worldwide allowing him to review it in his prime time which may not coincide with the sender's prime time. The recipient can then respond utilizing the same mechanism. The system queues and allows resend or rereceipt of messages while security prevents the transmitting of messages for which you are not the owner.

Benefits of TIOLR electronic filing for the secretary are the reduction of copying and messengering time. It also expands the word processor file size and makes the information created by word processors immediately and globally accessible to other users within the network. It reduces presentation cycle time by making the prepared documents im-

mediately available to all parties that are involved in the presentation preparation.

Benefits to the manager or professional are worldwide information availability and its timely distribution. Compatible versions of this system are available on both CIC and the TI Distributed Processors including support for online data transfer between Central and Distributed TIOLR files.

Benefits of the word processors are that they can transmit documents through TI's computer-communications network, act as an inquiry terminal and word processor at a secretary's work station, and significantly improve typing productivity.

#### SUMMARY

In summary, at TI we are now performing text management including local preparation via the word processor,

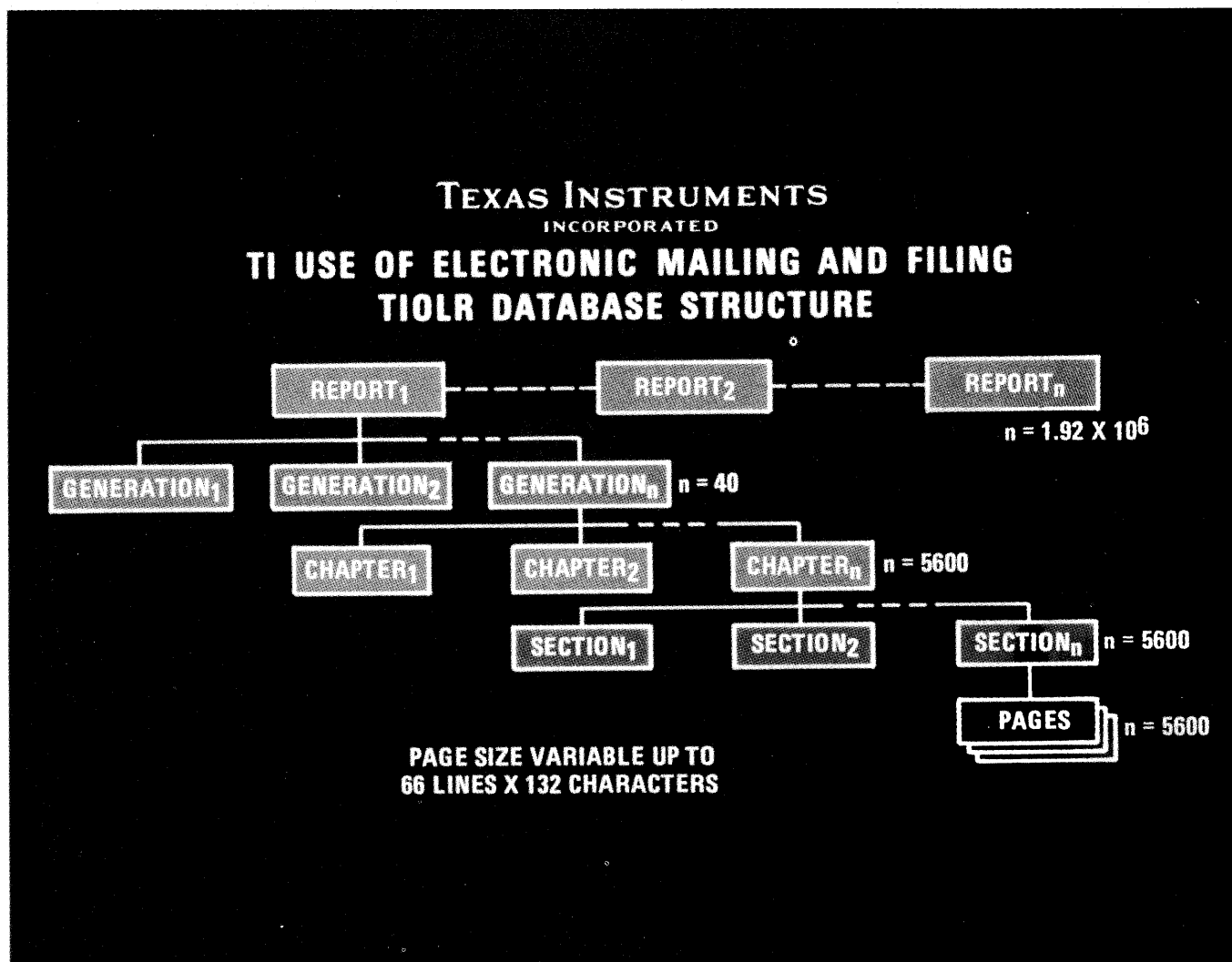


Figure 8



**TEXAS INSTRUMENTS**  
INCORPORATED

**TI USE OF ELECTRONIC MAILING AND FILING  
MSG VOLUME AND COST**

	YEAR END 1977	YEAR END 1978	AUGUST 1979
NUMBER OF MSG TRANSACTIONS PER DAY	10,500	16,300	19,800
CHARACTERS PER MESSAGE	444	437	442
RECIPIENTS PER MESSAGE	3	4	4
COST PER COPY RECEIVED	\$ .08	\$ .06	\$ .05

Figure 9

distribution via our MSG electronic mailing system, and generalized filing capability for storage and access via the TI On-Line Reporting system.

These capabilities are integrated with a full function computer-communications network in order to take advantage of our existing network. The word processors have been network connected in order to maximize the productivity gains for secretarial and clerical staff that utilize the word processor for local document preparation.

We are extending these word processing capabilities to our distributed processing systems in order to further maximize the benefits of an integrated word processing/data processing computer-communications network.

With the tools that we currently have in place we have realized significant productivity gains, however as these functional capabilities are enhanced and further deployed the benefits continue to accrue.

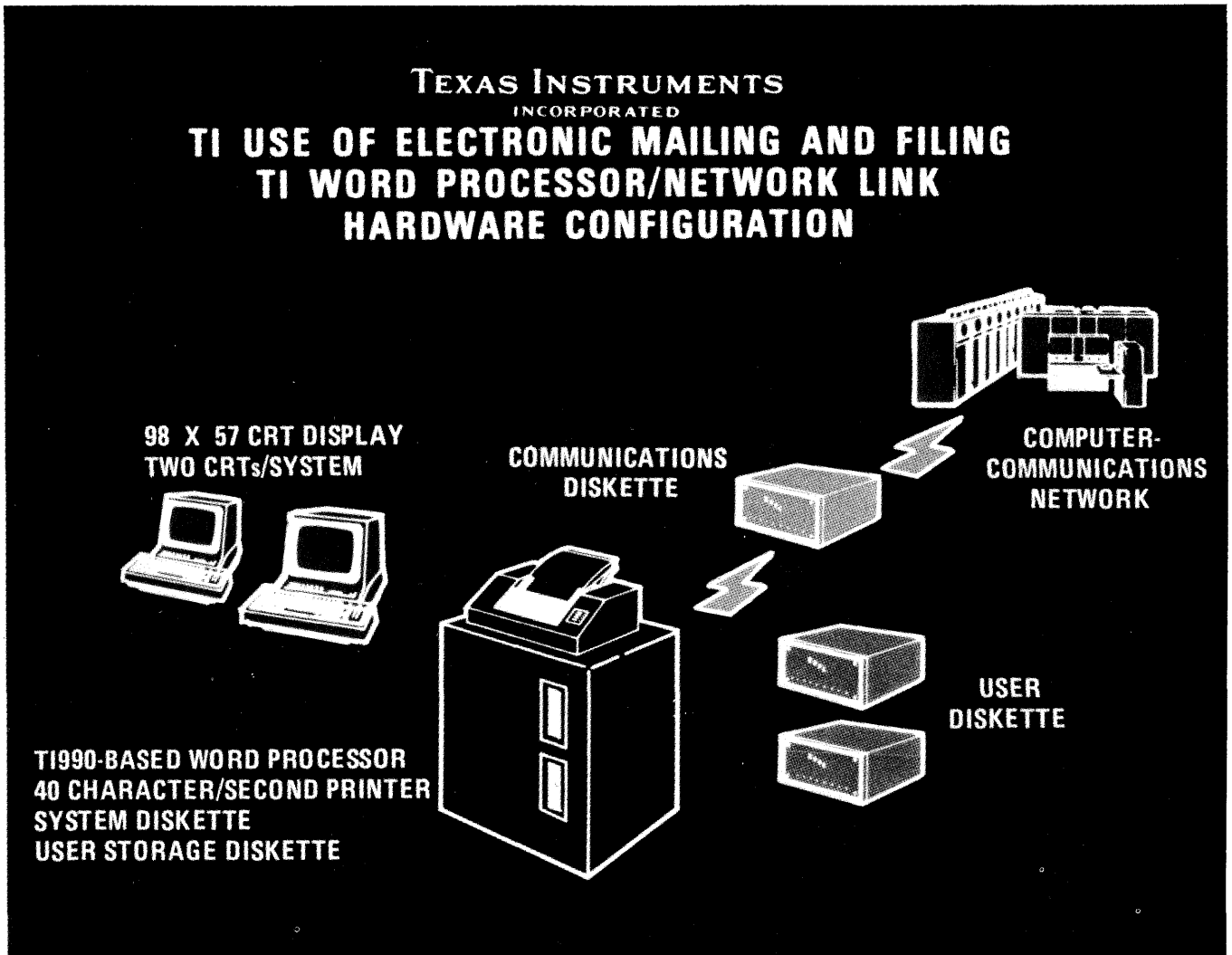


Figure 10

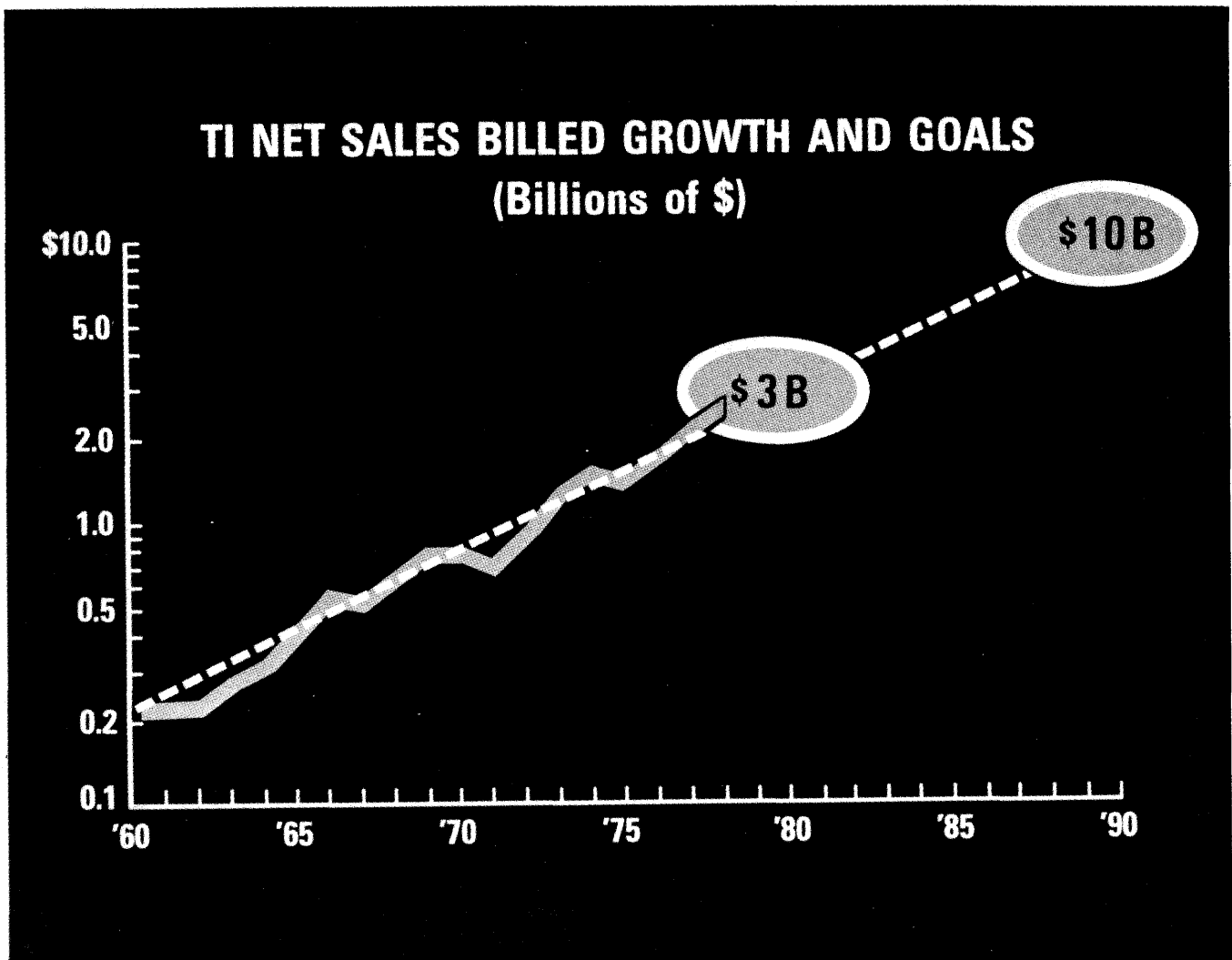


Figure 11

	<u>Y/E 77</u>	<u>Y/E 78</u>	<u>NOV. 79</u>
INQUIRY TERMINALS	2400	4900	8083
NETWORKED DISTRIBUTED PROCESSORS	39	70	135
COMMUNICATIONS PROCESSORS (TICOG)	19	28	32
STANDALONE DISTRIBUTED PROCESSORS (ESTIMATED)	3000	5000	7500

Figure 12—Information systems and services; TI distributed processing network growth.

# Implementing electronic mail in a telephone system: more than just talk

by GERALD TOMANEK

ROLM Corporation  
Santa Clara, California

## INTRODUCTION

For most of us, the office of the future is a vague concept. We can see some dim shapes through the mists which look like they might be the place we want to be in a few years, but there is no obvious road from here to there. It would help to define our goal so that we can know whether or not we have arrived at the right place once we have gotten somewhere, and it would help even more to know whether or not we are getting closer to it as we move forward. Then we could pick one of the paths that seems to lead in the right direction.

We are tempted to define the office of the future as a collection of hardware, software or procedures, but I believe it is really a label we give to a result we devoutly wish for: increased white collar productivity. We have many support systems today which look like paths toward that goal: data, text, image and the most pervasive information support system in the office, the telephone system. Each of our enterprises will find a different road to our own future office, a path that becomes clearer step by step as we proceed. For that reason, today's planning and purchasing decisions for these information systems must allow for this evolutionary, learning process. Flexibility, growth capacity, standard interfaces—these are the key words for the systems of today that can become the systems of the future.

In this piece, I will illustrate one of the growth paths by describing the evolution of a digital telephone system into an integrated voice and text information system. A specific type of electronic mail can now be provided from the telephone system to give the office workers a more complete information system, aimed at increasing their productivity. It also illustrates the potential future role of the digital telephone system as the brain and central nervous system of the office of the future. And it brings forward some of the organizational impact that will accompany the move into the future.

## WHITE COLLAR PRODUCTIVITY: THE NEED FOR NEW TOOLS

The office of the 1980's needs some new tools. The evidence is expressed in an increasingly familiar litany:

—White collar payroll has grown to more than 50 percent of total payroll.

—However, the productivity of white collar workers is increasing scarcely at all, up a mere 4 percent over a ten year period.

The search for solutions to this growing problem begins with an analysis of office worker occupations and the time spent in key activities.\* Sixty-six percent of white collar payroll dollars (not headcount) goes to managers, administrators and professionals (engineers, for example), 6 percent to secretaries and 28 percent to other clerical support workers. The first class of workers spends most of its time in information communication activities: gathering, sharing, creating and processing information in many ways. For a top executive, 95 percent of the day can be spent in these activities; for a design engineer, perhaps only 50 percent. But for this class as a whole, an average 75 percent of the day is spent reading, writing, making phone calls, going to meetings, preparing for meetings, even teleconferencing.

If most of the white collar payroll goes to workers who spend most of their time on information communications activities, the obvious place to look for increased productivity is the information support systems for these workers. Today we find word processing, the most advanced technology in the office, serving the secretarial workers, not helping a manager get significantly more done in a day. But it has given us a taste of what technology can do for us. The office of the 1980's must feature more information support systems, and they must be aimed primarily at the managers and professionals.

## AN ELECTRONIC MESSAGE SYSTEM (EMS)

One of the new information systems which can be introduced to the office for increased productivity is variously called an electronic message system, electronic mailbox or computer based message system. This system allows users

\* The figures cited are from "Communication in the Office of the Future: Where the Real Payoff May Be," James H. Bair, *Business Communications Review*, Jan.-Feb. 1979.

to send and receive message communications (those brief, informal and usually perishable intra-company communications) electronically. Messages are created, read and otherwise manipulated on terminals, which may be printers or paperless CRT's. After creation, messages are stored on online media for retrieval by the addressee. A central computer provides information control and processing functions for users of the system.

In practice, an electronic mailbox is used just as a physical mailbox. The user checks for new input a couple of times a day, discards or answers the messages, and creates new messages for other workers who use the system. Occasionally, he or she may want to forward a message to another worker for their information.

If terminals are conveniently located, an electronic message system can substitute for some of the phone calls to and from an office worker. All of those calls which are one way ("Monday's staff meeting is cancelled") or which don't require an immediate response ("Can we meet Thursday at 10:00?") can be handled by an electronic mailbox. Twenty to thirty percent of all intra-company calls may be of this nature. An EMS saves time otherwise spent in telephone tag, chasing back and forth to complete the communication.

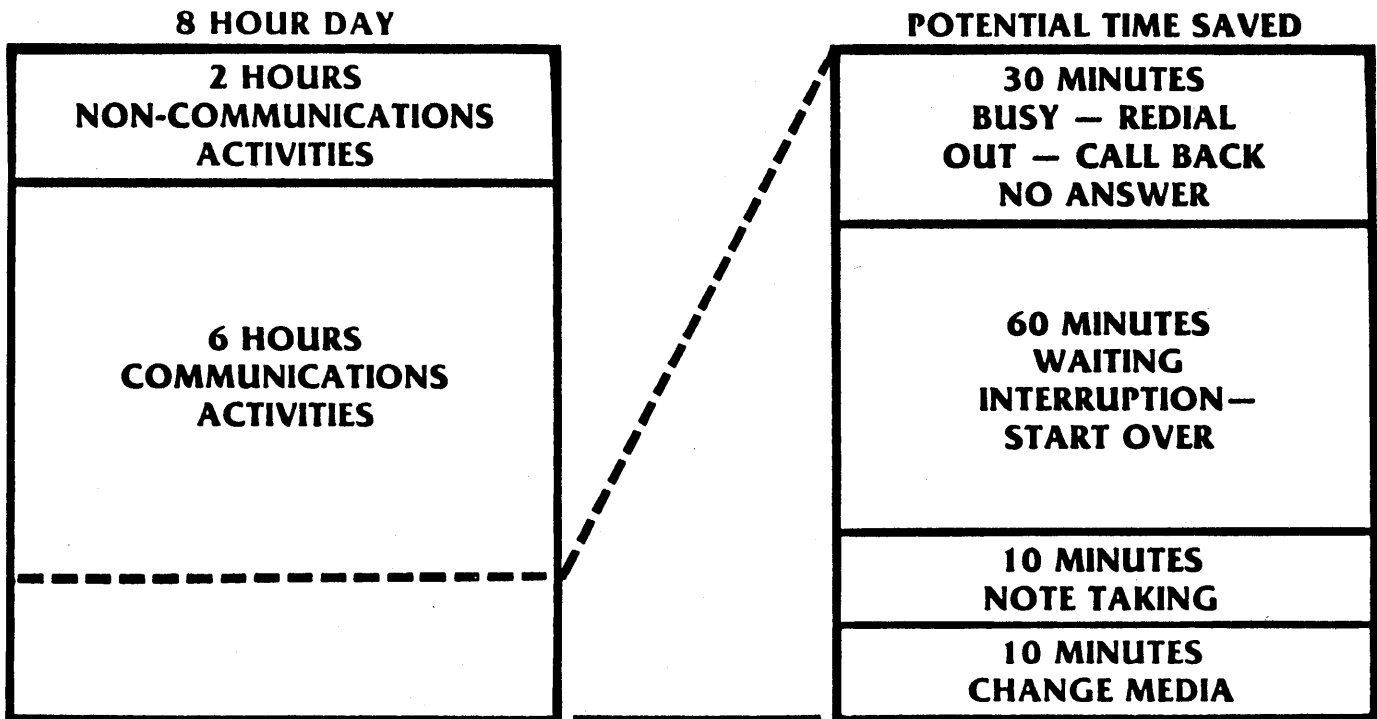
It also saves the time wasted when a phone call interrupts your dictation and you have to rewind your thinking, and the time wasted while another party searches for information. See Figure 1.

Electronic messages also substitute for some of the paper flow in the office. Because they are delivered to the electronic mailbox instantly, there is no expense for intra-company mail, copying costs, delivery costs, or postage. And instant delivery means no wasted effort due to the late arrival of an important message. It also means an increased feeling of teamwork among the participants, when all team members can be equally well informed, even if geographically spread out. The benefits compound if users are in multiple time zones, or travel frequently.

The elements needed to implement a successful electronic message system are:

- Input/output—Terminals should be convenient to the users, installed in or near their work area.
- Processor—Operation by many users at once is necessary, especially in larger offices. Response time must be short, on the order of 1 second or less, but the terminal input rate is small bursts at long intervals.

## THE PAY OFF — TIME SAVED FOR OFFICE WORKERS



SOURCE: SRI INTERNATIONAL

Figure 1—The pay off—time saved for office workers.

- Storage—For perishable messages, there must be enough online storage for several days' traffic. Many installations have found that the average user generates three 500 character messages a day, and that three days of storage is generally adequate. Each site and application has different requirements which alter these guidelines.
- Hard copy—Provision for hard copy is essential, although printers can be centralized or shared since their use is less frequent than CRT's.
- Simplicity—The system must be simple to operate by office workers with no technical or computer systems background; and it must provide prompting or assistance when necessary, since usage will be infrequent — once or twice a day with occasional lapses of several weeks.
- Remote access—Users who are not on site must be able to use the system to achieve the full benefits.
- Security—Mailboxes must be protected from access by other people.
- System management tools—The system manager must have statistics on usage and errors to effectively plan and control.

#### IMPLEMENTING AN EMS IN A TELEPHONE SYSTEM

There are many ways to implement an electronic message system. A mainframe or minicomputer can be programmed

to operate an electronic mailbox, and packaged software is available for this purpose. Time-sharing services also offer this capability. In 1979, ROLM Corp. announced a new way, integrating an electronic mailbox system with a telephone system, the ROLM CBX (Computerized Branch Exchange). This was called REMS, for ROLM Electronic Message System. See Figure 2.

The ROLM CBX is a computer controlled digital switch. To switch telephones, as a PABX, the analog signals are converted to digital form by pulse code modulation, and connected by a time division multiplex bus. The TDM bus is, of course, indifferent to whether the digits being switched originated from telephones or business equipment such as terminals.

To interface terminals to the TDM network, ROLM developed an RS232 interface for its electronic telephone set, the ETS 100™. See Figure 3. This telephone set has a pair of wires connecting its analog voice signal to the switch, and another pair carrying data to and from the switch for signalling and supervision. With the addition of the terminal interface, terminal data is combined with signalling data, and the result is an integrated voice and data communication system using common telephone wiring without modems. Furthermore, the addition of the data interface does not detract from telephone service, an important point since even in the office of the future, voice will remain a major form of communication. Telephone and terminal may be used in-

## SYSTEM DIAGRAM

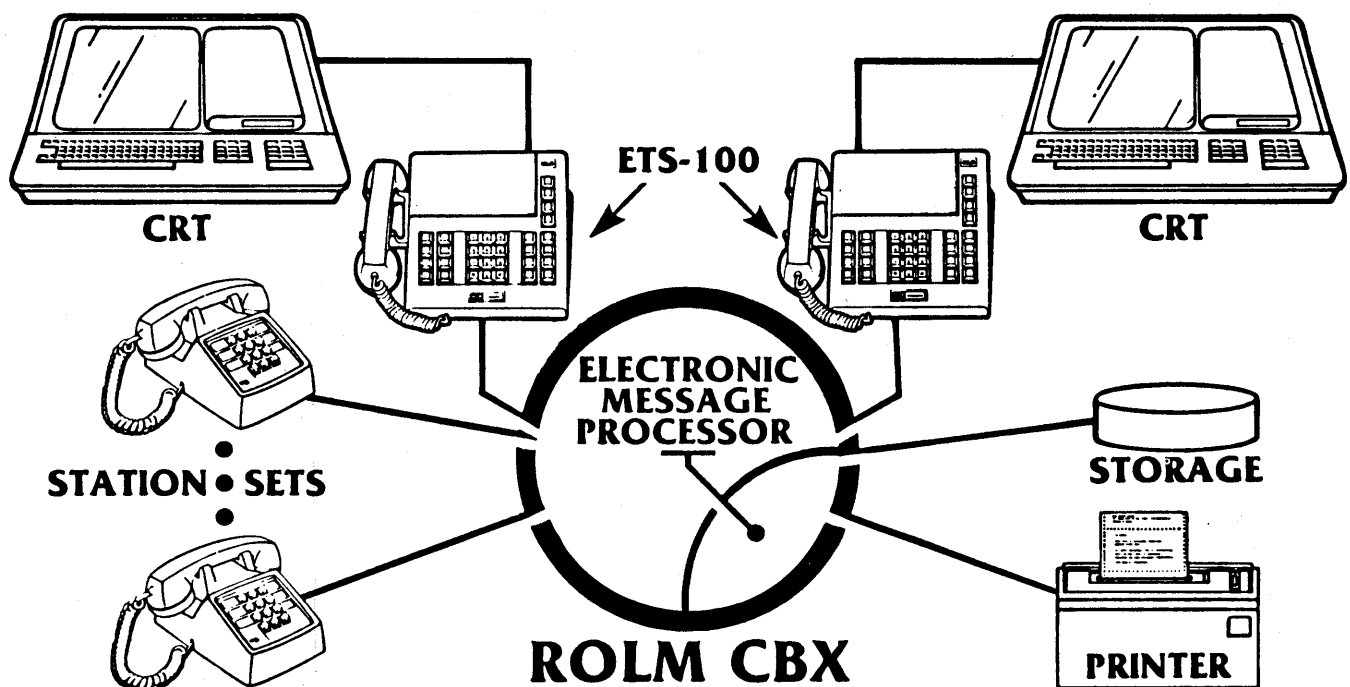


Figure 2—System diagram.

# MESSAGE TERMINAL PLUGS INTO ETS-100

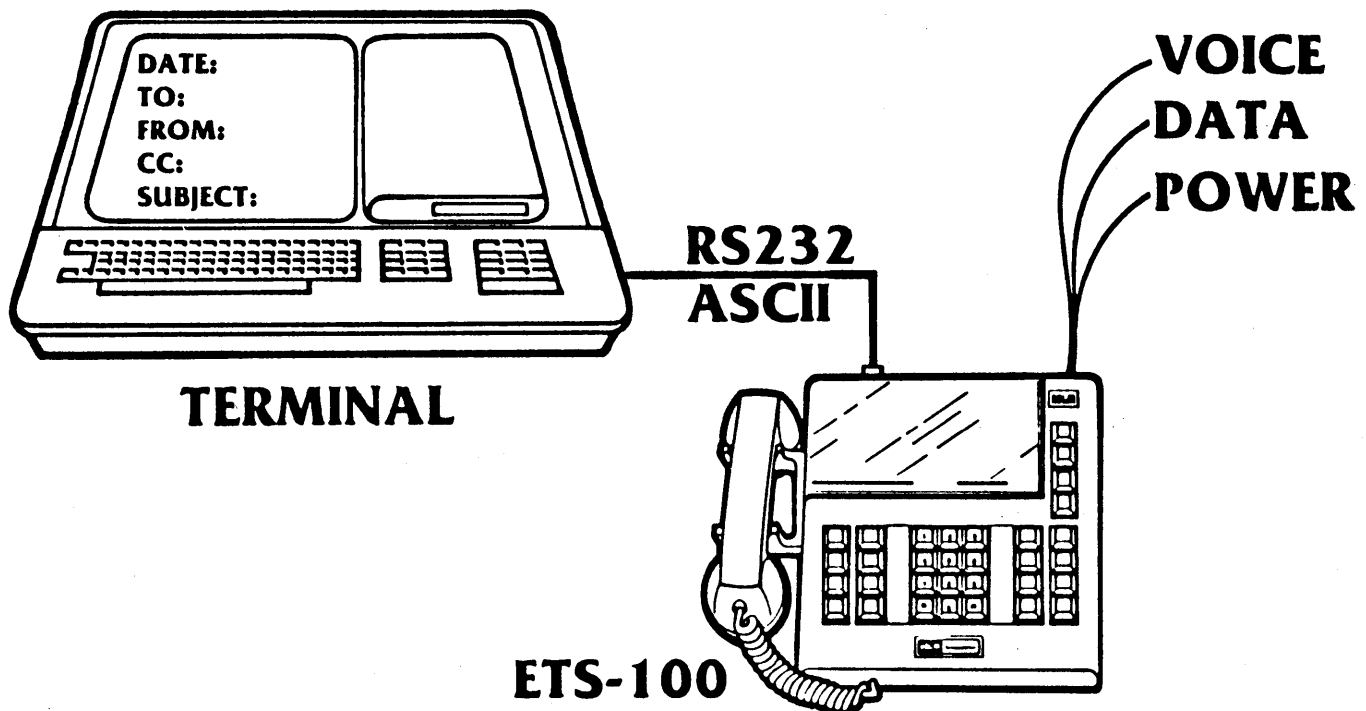


Figure 3—Message terminal plugs into ETS-100.

dependently, simultaneously without affecting each other's performance.

The processor to operate REMS was easily found. The CBX is run by a 16 bit minicomputer, and a CBX can be configured to have redundant processors, one maintained on standby processor with the electronic message system software if the primary processor should fail. By programming this standby processor with the electronic message system software, ROLM integrated REMS in the CBX. Data from a general class of ASCII, non-buffered terminals goes to the CBX over telephone wiring, onto the TDM bus where several simultaneous users can be accessing the system running in the standby processor.

Storage of up to 1 MB for messages is provided on floppy disks. For average use, this storage is adequate for 200 users. Hard copy can be produced at a printer nearest the author or recipient of a message on command, and REMS allows the use of CRT's with integral or daisy chained printers for further hard copy options.

Remote access to REMS is possible via dial up line using a terminal and modem. A modem at the CBX allows the

remote terminal to be connected to the REMS processor, and thereby use REMS identically to all on-site terminals.

REMS can deliver messages to remote terminals, too, using the modem at the CBX. With this capability, a complete store and forward message system can be constructed.

The commands to operate REMS are simple English language words (SEND, READ, PRINT, etc.) which require no typing skill to peck out. Prompting is available, HELP can be requested, and error messages are provided so that even infrequent users can operate the system. Proficient users can elect to use an abbreviated set of commands, but drop back to the basic level if necessary.

Security is provided by password access which can be changed by the user from the terminal. Indeed, an electronic mailbox is more secure than the typical inbox. In addition, remote terminals in open areas can be configured not to print a message but rather a notice that a message is waiting to be read.

Finally, REMS provides the system manager with statistics on system usage, memory, and errors for planning and control.

## THE ADVANTAGE OF AN EMS IN A TELEPHONE SYSTEM

The advantages of integrating an EMS in a CBX are both economic and enhanced performance. The most obvious advantages are the elimination of modems and special wiring, a single vendor for both voice and message systems, and usually the lower cost of the total system. Since the CBX purchase is justified on the basis of cost reductions in voice communications alone, typically 33 percent, REMS can be viewed as an incremental expansion of the CBX, costing less than a separate stand-alone system.

Integrating REMS in the CBX provides features and performance which a stand-alone system could not offer without considerably more expense. For example, a message alert can be provided to the user through the telephone system, notifying him that a new message has arrived in his mailbox via a display on his electronic telephone, or potentially in other ways. Message delivery can take advantage of the least cost route selection which the CBX can perform for all outgoing calls. And users gain a great deal of flexibility by using the telephone wiring. Terminals can be added easily, and relocated without expensive rewiring.

## FUTURE DEVELOPMENT

The future of REMS will depend on users' experiences with it. Along the evolutionary path to the office of the future, as users discover what they like and dislike, the system will develop accordingly. Software will be modified and hardware added in the likely direction of more functionality at the terminal, more network options, interfaces or integration with other systems, more storage, or specific applications such as nurses' stations.

REMS illustrates the potential future development of information systems integrated in a CBX in several ways. The

use of the telephone wiring as an office central nervous system can certainly be applied to many applications, and the switching, concentrating and information processing functions of the CBX can be extended to provide many information services to the office workers.

## ORGANIZATIONAL IMPACT

The impact on the organization has to be considered when any move toward the future office is evaluated. Unfortunately, many proposed solutions require significant changes, such as giving up secretaries or changing job descriptions, to deliver their benefits. Consequently, they are slow to be accepted.

An EMS requires no significant organizational change to deliver its benefits. For the information services manager looking for an opportunity to begin moving toward the future office, an EMS is a non-threatening supplement to the existing information systems, but it provides an opportunity to introduce office workers to terminals, electronic mail, store and forward systems and electronic filing and retrieval.

A company should be prepared to recognize and reward the office workers who succeed in using the new tools to produce more results. Most firms are unlikely to lay off their managers and professionals, even if new systems succeed in freeing time in the day. The additional results that these people can produce with that time have far more impact on the bottom line than payroll cost reduction, but the organizational climate has to favor that behavior.

An interesting organizational impact of an EMS is the opportunity it provides for a firm's information services managers to work together. The voice communications and office automation managers can use the installation of an EMS to establish common goals and objectives, develop specifications, and learn to work as a team to deliver the new system to their users. The future office will come that much nearer as they combine their imaginations and talents.





# An office form flow model

by IVOR LADD and D. C. TSICHRITZIS

University of Toronto  
Toronto, Ontario

## INTRODUCTION

Offices are data processing systems involving complex man-machine interactions. In traditional offices, machines have played a passive role. They aid in organizing, preparing, copying, storing, transmitting, analyzing and transforming data, but operations are initiated and directed by people. Replacing machines by computer systems for word processing, phototypesetting, database management and electronic mail may increase the efficiency of the offices, but it does not change their passive role. In automated offices, computer systems will be designed to play a more active role. Many well-defined routine operations can be initiated and directed by computer systems. People can then concentrate on more challenging tasks.

The effectiveness of automated offices depends largely on the success of formally describing and analyzing the well-defined portions of traditional offices. The need for formal descriptive and analytic tools gives rise to the study of formal models of offices.<sup>4</sup> The diverse aspects of offices lead to different modeling approaches. The models of Ellis and Zisman<sup>3,9</sup> focused on the description, precedence and synchronization of tasks or activities within an office. The model of Tschritzis<sup>8</sup> focused on the description and analysis of the office structure and components. The form flow model presented in this paper follows the latter approach.

The form flow model (FFM) regards an office as a network of stations through which forms flow. Forms are structured data. Stations are processing units. The network coordinates the routing of forms between stations. Thus forms originate in some initial stations of the network, flow from station to station where they are processed, and terminate in some final stations.

Consider Figure 1, an example of a simplified loan processing office. It consists of five stations: a receptionist (R), a processing clerk (PC) and his manager (PM), and a credit clerk (CC) and his manager (CM). R is given all the loan application forms. He considers those with invalid contents to be rejected overall and sends the rest to PC. PC first sends those from low income applicants to PM for approval. Then he sends the remaining forms and the forms returned by PM to CC. CC first sends those of large loans to CM for approval. Then he considers the remaining forms and the forms returned by CM to be approved overall. PM and CM either approve or reject the forms they receive. They return

the approved forms to PC and CC respectively and they consider the rejected forms to be rejected overall.

## DEFINITION

Each form in the form flow model contains fields for holding data values. An individual form (instance) carries three components  $(\langle \zeta_T, \zeta_K, \zeta_C \rangle)$ : its form type ( $\zeta_T$ —the kind of fields it contains), its form key ( $\zeta_K$ —a permanent unique identifier) and its form contents ( $\zeta_C$ —the field values).

Each station ( $s_i$ ) in the form flow model has a set of in trays ( $T_{Ii}$ ) and a set of out trays ( $T_{Oi}$ ) where forms are deposited. The task of a station is to take a form  $(\langle \zeta_T, \zeta_K, \zeta_C \rangle)$  from an in tray ( $x \in T_{Ii}$ ), apply an operation selection function associated with the in tray ( $r_x: \{\langle \zeta_T, \zeta_K, \zeta_C \rangle\} \rightarrow T_{Oi}$ ) to determine to which out tray ( $y \in \text{range}(r_x)$ ) the form is to be transferred, perform an operation associated with the in tray-out tray pair ( $a_{xy}: \{\langle \zeta_T, \zeta_K, \zeta_C \rangle\} \rightarrow \{\langle \zeta_T, \zeta_K, \zeta_C' \rangle\}$ ), and deposit the transformed form  $(\langle \zeta_T, \zeta_K, \zeta_C' \rangle)$  in that out tray ( $y$ ). A flow arc  $((x, y) \in E_{Ai})$  exists between an in tray ( $x$ ) and an out tray ( $y$ ) if it is potentially possible for the operation selection function associated with that in tray to select that out tray ( $y \in \text{range}(r_x)$ ). Since each operation flow arc represents an operation, depending on the analysis, costs (times, weights, capabilities, etc.) may be associated with the flow arcs ( $E_A = \cup_i E_{Ai}$ ). It may be desirable to abstract out operation selection functions by estimating the frequencies of selecting the different flow arcs in the network.

The network in the form flow model designates a subset of all in trays ( $T_I = \cup_i T_{Ii}$ ) as initial trays ( $T_\alpha \subseteq T_I$ ) where forms originate in the network. The network also designates a subset of all out trays ( $T_O = \cup_i T_{Oi}$ ) as final trays ( $T_\omega \subseteq T_O$ ) where forms terminate in the network. The task of the network is to take a form  $(\langle \zeta_T, \zeta_K, \zeta_C \rangle)$  from a non-final out tray ( $y \in T_O - T_\omega$ ), apply a routing selection function associated with the out tray ( $r_y: \{\langle \zeta_T, \zeta_K, \zeta_C \rangle\} \rightarrow T_I$ ) to determine to which in tray ( $x \in \text{range}(r_y)$ ) the form is to be transferred, and deposit the form in that in tray ( $x$ ). A flow arc  $((y, x) \in E_C)$  exists between a non-final out tray ( $y$ ) and an in tray ( $x$ ) if it is potentially possible for the routing selection function associated with that out tray to select that in tray ( $x \in \text{range}(r_y)$ ). Since each routing flow arc represents a communications link between two stations, depending on the analysis, costs may be associated with the flow arcs. It may

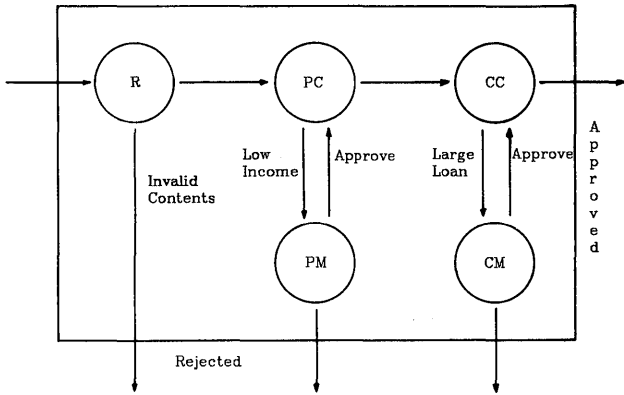


Figure 1—Loan processing office.

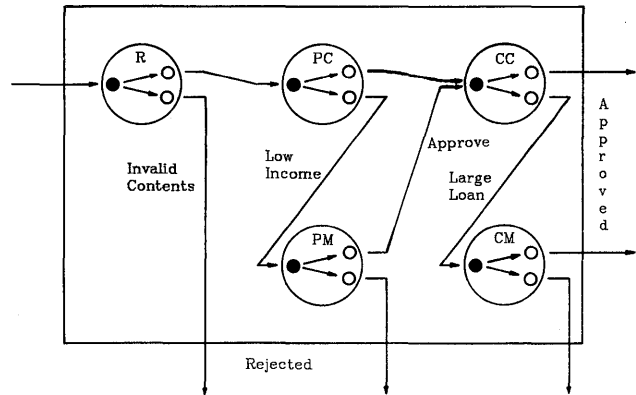


Figure 3—FFM of simpler loan processing office.

be convenient to abstract out the costs by absorbing them in the operation flow arcs. Routing selection functions can in fact be modeled by operation selection functions (see appendix). Furthermore, it may be desirable to abstract out routing selection functions by estimating the frequencies of selecting the different flow arcs in the network.

By the above definition of the form flow model, forms are conserved in the network. They are neither created nor destroyed, only transformed. Hence the model is a true flow network.

Consider Figure 2, an equivalent of Figure 1. The in trays, out trays, operation flow arcs and routing flow arcs are explicitly shown here. Consider Figure 3, a simpler but functionally equivalent version of Figure 2. It will be used to illustrate subsequent analyses. Here, PM sends the forms he approves directly to CC instead of through PC. CM does not return forms to CC, but considers them to be approved or rejected overall himself. This simplification is an indication of the optimization which can be obtained from the form flow model, but which needs to be formalized.

The form flow model and equivalent models having only operation or routing selection functions are formally defined in the appendix.

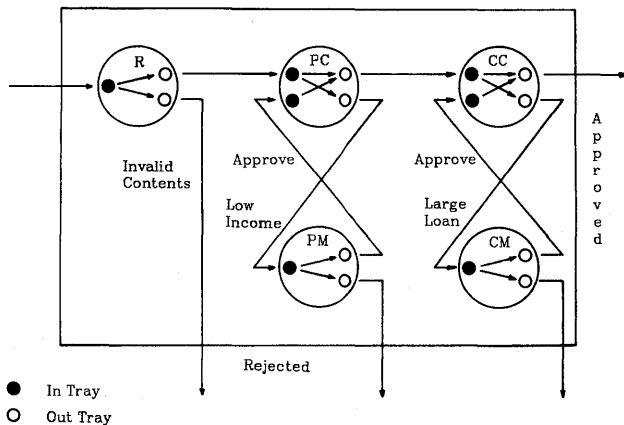


Figure 2—FFM of loan processing office.

### RESTRICTIONS

A number of restrictions can be placed on the form flow model to simplify it. The first restriction is for the network to be deterministic. This means that the sequence of operations performed on a form from its initial state to its final state should result in a deterministic transformation of the form contents. Note that this does not mean that the sequence of operations, the operation and routing selection functions, or the operations themselves must be deterministic. As long as sequences of operations can be partitioned into equivalence classes and any non-determinisms do not cause sequences to cross equivalence classes, then an overall deterministic behavior can be defined. It is desirable to allow the routing selection functions to be non-deterministic. In a situation where two or more stations (for example, typists) perform essentially the same operations, it may not matter to which of the stations a form is routed. Hence one of the stations should be selected non-deterministically. In another situation where two operations are commutative or independent, it may not matter which of the operations is performed first. Hence the performance of the operations should be possible in either order non-deterministically. There are fewer realistic situations where non-deterministic operation selection functions and operations are desired and where the non-determinism cannot be limited to the routing selection functions. However, deterministic operation selection functions and operations may be so complicated that they can only be treated reasonably as being non-deterministic. In any case, for the sake of simplicity, only the routing selection functions are assumed to be non-deterministic.

The second restriction is for the network to be isolated from all factors external to it. This means that all external factors are ignored. External data are not referenced unless they are explicitly included in the contents of an inputted form. External actions are not triggered except by indicating the actions in the contents of an outputted form. Furthermore, to make analysis more tractable, operation and routing selection functions and operations are assumed to be isolated from each other and the current state of the network (how many forms are in the network? where? etc.). However, it is often desirable to allow routing selection functions to be

non-isolated from the state of the network. In a situation where a form may be routed to either of two equivalent stations, one optimizing heuristic is to route the form to the less busy station. To do this, the particular routing selection function must be aware of the state of the relevant stations.

The third restriction is for the network to be memoryless. This means that the stations cannot record form contents for future reference. Stations are allowed to have memory and to record the contents of a form as it passes through them, but they are also required to erase all data recorded from that form after it is outputted from the network.

With the above three restrictions on the form flow model, the network is deterministic, isolated and memoryless and is thus a function of form contents. Moreover, as stations are transformations on form contents, the network can then be mapped into a station and the form flow model becomes hierarchically decomposable. Consequently, a complicated form flow model can be partitioned. Each sub-network can be analyzed separately and then mapped into a station. Finally, the mapped stations can be reassembled into a simpler network for analysis.

GRAPH THEORETIC ANALYSIS

For the graph theoretic approach, the form flow network can be considered as a directed graph of in trays and out trays and flow arcs (G(T,E)). A flow path is a sequence of incident flow arcs leading from an initial tray to a final tray. Each flow path defines a sequence of stations. Each flow path also defines a sequence of operations. However, the sequence of operations may violate the consistency constraints on the selection functions and operations. Hence a flow path may never be followed by a form in actual processing. As it is often desirable to suppress the details of the selection functions and operations, one of the goals of this analysis is to achieve as close a correspondence as possible between the flow paths and the paths which may be followed by a form in actual processing (followable flow paths), or in other words to avoid having flow paths which are not followed. The use of multiple in and out trays in stations can help to some extent to attain this goal. In Figure 2, all flow paths can be followed. Figure 4 may be made equivalent to Figure

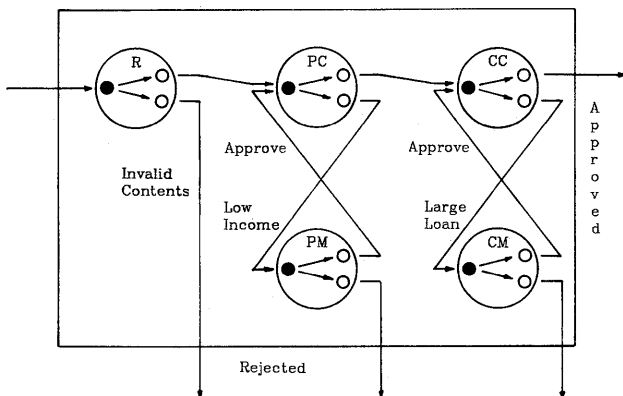


Figure 4—FFM with loops.

2 with appropriate selection functions which restrict certain flow paths such as the repetitions of the loops between PC and PM and between CC and CM. Thus, for the sake of simplicity, it will be assumed that the flow paths in all figures except Figures 1 and 4 can be followed.

With the concept of followable flow paths arises the related concept of an infinite sequence of low arcs leading from an initial tray. If an infinite sequence exists and is followable, then a form can be trapped forever in the network. This situation usually indicates an error in the formulation of the network and is to be avoided. An infinite sequence in a finite network implies a loop of flow arcs. Hence, in a network where all flow paths are followable, the existence of a loop indicates an error. In a network where all flow paths are not necessarily followable, the existence of a loop indicates an error only if there is no upper bound on the length of the followable flow paths.

The flow paths in a network can be enumerated with a depth first search (DFS) algorithm. At the same time, the frequency and average processing time of following each flow path can be computed given the frequency and average processing time of following each flow arc. Let  $q(x,y)$  where  $(x,y) \in E$  and  $t(x,y)$  where  $(x,y) \in E_A$  be the frequency and average processing times of following the arc  $(x,y)$ . Let

$$p(x,y) = q(x,y) / \sum_{\substack{y' \in T_A \\ (x,y') \in E}} q(x,y')$$

The following algorithm performs the computation.

```

DFS(x, seq, time, freq)
{
  if(x ∈ seq)
    found_loop;
  ∀y ∈ To(x,y) ∈ EA
  {
    seq' ← seq, (x,y);
    time' ← time + t(x,y);
    freq' ← freq × p(x,y);

    If(y ∈ T∞)
      print seq', time', freq';
    else ∀x' ∈ Ti(y,x') ∈ EC
      DFS(x', seq', time', freq' × p(x,y));
  }
}
∀x ∈ Tα
DFS(x, φ, 0, 1);
    
```

Consider Figure 5 an equivalent of Figure 3 for graph theoretic analysis. Frequency and time parameters were estimated to reflect a reasonable situation. Applying the DFS algorithm to it produced the following:

Flow Path	Time	Frequency
R <sub>1</sub> PC <sub>1</sub> CC <sub>1</sub>	5	.2160
R <sub>1</sub> PC <sub>1</sub> CC <sub>2</sub> CM <sub>1</sub>	14	.0270
R <sub>1</sub> PC <sub>1</sub> CC <sub>2</sub> CM <sub>2</sub>	13	.0270
R <sub>1</sub> PC <sub>2</sub> PM <sub>1</sub> CC <sub>1</sub>	12	.2520
R <sub>1</sub> PC <sub>2</sub> PM <sub>1</sub> CC <sub>2</sub> CM <sub>1</sub>	21	.0315
R <sub>1</sub> PC <sub>2</sub> PM <sub>1</sub> CC <sub>2</sub> CM <sub>2</sub>	20	.0315
R <sub>1</sub> PC <sub>2</sub> PM <sub>2</sub>	8	.3150
R <sub>2</sub>	1	.1

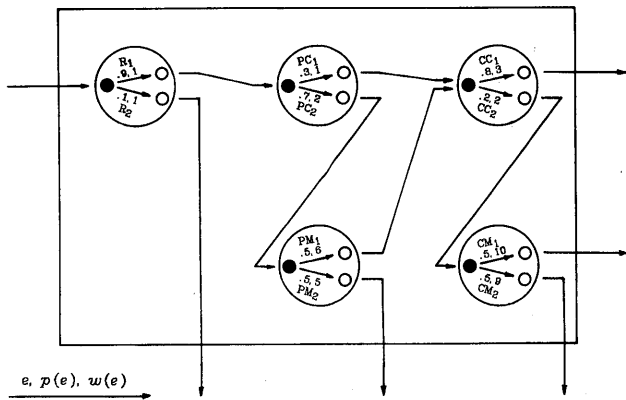


Figure 5—FFM for graph theoretic analysis.

As there may be many flow paths in a network, it is often convenient to group them into equivalence classes. Then, characteristics can be discussed in terms of a small number of classes as opposed to a large number of flow paths. Different notions of equivalence classes can be defined.

One notion of equivalence classes is grouping by initial tray-final tray pairs. This emphasizes the input-output aspects of the network as a whole and is useful for mapping the network into a single station. In Figure 5, the classes are: (rejected by R), (rejected by PM), (rejected by CM), (approved by CC), and (approved by CM). The class (approved by CC) combines the flow paths  $R_1PC_1CC_1$  and  $R_1PC_2PM_1CC_1$ , the class (approved by CM) combines the flow paths  $R_1PC_1CC_2CM_1$  and  $R_1PC_2PM_1CC_2CM_1$ , and so on.

The second notion of equivalence classes is grouping by the sequence of stations. This emphasizes the flow aspects of the network through the stations and is useful for analyzing the characteristics of the stations. For the example in Figure 6, the classes are: (R), (R,PC,PM), (R,PC,CC), (R,PC,PM,CC), (R,PC,CC,CM) and (R,PC,PM,CC,CM). The class (R,PC,CC,CM) combines the flow paths  $R_1PC_1CC_2CM_1$  and  $R_1PC_1CC_2CM_2$  and the class (R,PC,PM,CC,CM) combines the flow paths  $R_1PC_2PM_1CC_2CM_1$  and  $R_1PC_2PM_1CC_2CM_2$ .

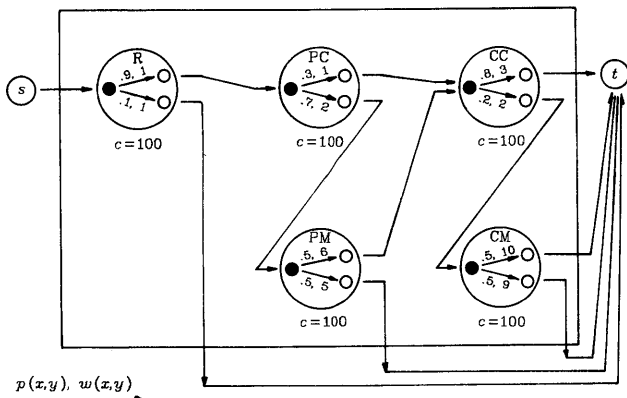


Figure 6—FFM for commodity flow analysis.

The third notion of equivalence classes is grouping by the equivalence classes of the sequences of operations. This emphasizes the processing aspects of the network and is useful for determining the transformations of form contents. For the example in Figure 6, the classes are: (rejected by R), (rejected by PM), (rejected by CM), (approved by CC), (approved by CM), (approved by PM—approved by CC), (approved by PM—approved by CM) and (approved by PM—rejected by CM). Here the flow paths correspond one-to-one with the classes for each every flow arc corresponds to a unique operation.

COMMODITY FLOW ANALYSIS

For the commodity flow approach, the form flow network can be considered as a commodity flow network (CFN). It can then be augmented to analyze its capacity in terms of maximum flow.<sup>5</sup> A source node ( $s$ ) from which arcs are directed to all the initial trays and a sink node ( $t$ ) to which arcs are directed from all the final trays are added. Formally,  $CFN(N,E)$  is constructed from FFM as follows.

$$N = TU\{s,t\}.$$

$$E = FFM(E) \cup \{(s,x) | x \in T_\alpha\} \cup \{(y,t) | y \in T_\omega\}.$$

A flow in CFN is a function  $f: E \rightarrow (0, \infty)$  satisfying the following constraints.

$$\forall e \in E, f(e) \geq 0.$$

$$\forall x \in N, \sum_{\substack{p \in N_\alpha \\ (p,x) \in E}} f(p,x) - \sum_{\substack{q \in N_\omega \\ (x,q) \in E}} f(x,q) = \begin{cases} -V(f) & \text{if } x = s, \\ +V(f) & \text{if } x = t, \\ 0 & \text{otherwise.} \end{cases}$$

The problem is to maximize  $V(f)$  subject to constraints on the capacities of the stations. Each station ( $s_i$ ) is assigned a value ( $c_i$ ) indicating the maximum number of work units it can perform per unit of time. Each operation flow arc  $((x,y) \in E_{Ai} \& x \in T_{ii})$  in the station is assigned a weight ( $w(x,y)$ ) indicating the number of work units associated with the work done by taking the arc. Then the constraint can be added:

$$\forall i, \sum_{e \in FFM(E_{Ai})} w(e) \times f(e) \leq c_i.$$

As this type of constraint is not considered in classical solution techniques for maximum flow problems, the network must be solved by linear programming techniques,<sup>2</sup> for example the simplex method.

Consider Figure 6, an equivalent of Figures 3 and 5. The capacity parameters from Figure 5 were used. Solving this network gave a maximum flow of 100 units. However, this flow is dominated by rejections by R and is not meaningful. It is necessary to include a constraint on the frequency of taking each arc. Furthermore, as forms from different initial trays may compete for flow through a station, it may be necessary to include also a constraint on the frequency of

input to each initial tray. Let  $q(x,y)$  where  $(x,y) \in \text{FFM}(E) \cup \{(s,x) | x \in T_\alpha\}$  be the frequency of taking arc  $(x,y)$ . Let

$$p(x,q) = q(x,y) / \sum_{\substack{y' \in N_\alpha \\ (x,y') \in E}} q(x,y')$$

Then the constraint is

$$\forall (x,y) \in \text{FFM}(E) \cup \{(s,x) | x \in T_\alpha\}, f(x,y) = p(x,y) \times \sum_{\substack{n \in N_\alpha \\ (n,x) \in E}} f(n,x).$$

Solving the network with this constraint gave a maximum flow of 28.86 units. For each station  $s_i$ , the value  $w_i = \sum_{e \in \text{FFM}(E_A)} w(e) \times f(e)$  gives the amount of work done by the station. The ratio  $w_i/c_i$  gives the utilization of the station. For the maximum flow, the values were:

station	$w_i$	$w_i/c_i$
R	28.86	.29
PC	44.15	.44
PM	100.00	1.00
CC	47.27	.47
CM	32.07	.32

This shows that PM is the bottleneck.

### QUEUING NETWORK ANALYSIS

For the queuing network approach, the form flow network can be considered as a queuing network to analyze its performance in terms of equilibrium behavior. For this purpose, the  $\text{FFM}_C$  (see appendix), an equivalent model of the form flow network, will be used. The  $\text{FFM}_C$  has only the routing selection functions and its stations comprise two parts, one for routing, the other for operations. It may be viewed as a multi-class open queuing system. Each form becomes a job. Each station becomes a server with a single queue corresponding to its set of in trays. Each operation in a station is associated with a different job class. To parameterize the system, a scheduling discipline for each station, a service time distribution for each operation and an interarrival time distribution for each initial tray must be specified. Furthermore, the routing frequencies which correspond to the frequencies of taking the routing flow arcs must be given. Since allowing state-dependent routing frequencies corresponds to allowing nonisolated routing selection functions, for the sake of simplicity, routing frequencies are assumed to be state-independent.

Formally,

- $S = \text{FFM}_C(S)$  are the servers;
- $C = E_A^C \cup E_A^A$  are the classes;
- $D: S \rightarrow \{\text{scheduling discipline}\}$ ;
- $X: C \rightarrow \{\text{service time distribution}\}$ ;
- $A: T_\alpha \rightarrow \{\text{interarrival time distribution}\}$ ;
- $Q: E_C \rightarrow (0, \infty)$  are the routing frequencies.

The remaining parameters can be computed.

$A_C: S \times C \rightarrow \{\text{interarrival time distribution}\}$  are the interarrival time distributions to particular classes and are defined by:

$$A_C(i,c) = \begin{cases} A(x) & \text{if } c = (x,x') \in E_A^C \text{ \& } x \in T_\alpha \\ 0 & \text{otherwise.} \end{cases}$$

$$P(y,x) = Q(y,x) / \sum_{\substack{x' \in T_\alpha \\ (y,x') \in E_C}} Q(y,x')$$

where  $(y,x) \in E_C$  are the routing probabilities.  $P_C: S \times C \times S' \times C' \rightarrow [0,1]$  are the class transition probabilities and are defined by:

$$P_C(i,c,i',c') = \begin{cases} P(y,x') & \text{if } c = (x,y) \text{ \& } c' = (x',y') \text{ \& } (y,x') \in E_C \\ 0 & \text{otherwise.} \end{cases}$$

To simplify the system, the routing servers  $\{(s^c | s^c \in \text{FFM}(S))\}$  can be assumed to operate in zero time and not to need scheduling disciplines and service time distributions.

The queuing network can be solved exactly by analytic techniques provided some further assumptions and restrictions are made.<sup>7</sup> Otherwise approximation techniques or simulation must be used. Three assumptions are necessary to have an exact solution: one-step behavior, server homogeneity and routing homogeneity. One-step behavior means that the only observable changes in a system result from single jobs either entering the system, flowing from one server to another, or leaving the system. Server homogeneity means that the service time distributions of a server may depend only on the state of its queue. Both these assumptions are reasonable for an office. Routing homogeneity means that the routing frequencies may depend only on the total number of jobs currently in the system. This assumption is satisfied by the above specification of constant routing frequencies.

Although an exact solution is possible with the above assumptions, a fast solution technique<sup>1</sup> is known only for cases with certain combinations of the three parameters: interarrival time distribution, scheduling discipline and service time distribution. The interarrival time distributions are restricted to be exponential. The scheduling disciplines are restricted to be FCFS (first come first served), PS (processor sharing simultaneously among all jobs in the queue), NQ (no queuing—the server supplies as many processors running at the full rate as there are jobs in the queue), or PLCFS (pre-emptive last come first served). If the scheduling discipline for a server is FCFS, then all service time distributions for the server are restricted to be identically exponential. If the scheduling discipline for a server is PS, NQ, or PLCFS, then each service time distribution for the server is restricted to having a rational Laplace transform. The above restrictions are severe and not entirely satisfactory since in an office, FCFS with different service time distributions or HOL (head of line—FCFS for each class with non-pre-emptive priority among different classes) seem more realistic. Nevertheless, to solve a queuing network at present, the restrictions must be imposed.

Consider Figure 7, an equivalent of Figures 3, 5 and 6. The parameters were taken Figure 6 to reflect near saturation

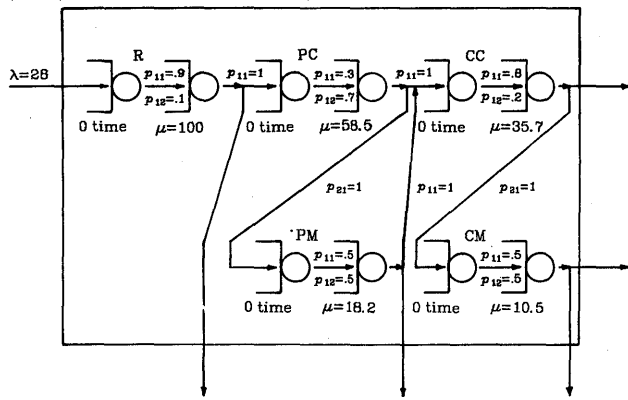


Figure 7—FFM for queuing network analysis.

condition as computed with maximum flow analysis. The interarrival time distribution is exponential with parameter  $\lambda$ . The scheduling discipline for each server  $s$  is FCFS and the service time distributions for the server are exponential with parameter  $\mu_s$ . The results of solving the network are given below.

Server Throughput Mean Queue Length Utilization

Server	Throughput	Mean Queue Length	Utilization
R	28.0	.39	.28
PC	25.2	.75	.43
PM	17.6	31.50	.97
CC	16.4	.85	.46
CM	3.3	.45	.31

Note that the utilizations here correspond closely to the values  $w_i/c_i$  for maximum flow analysis.

CONCLUDING REMARKS

We are still developing the form flow model. The types of analyses discussed above can be extended. The semantics of equivalence classes of flow paths and other graph properties need further investigation. Solution techniques for commodity flow and queuing networks as applied to form flow networks need extensions to less restrictive cases and improvements in speed.

New types of analyses can be studied. The problem of optimization is appealing. Suitable restructuring rules need to be formalized and cost functions need to be defined. Then, algorithms may be developed to derive optimal or semi-optimal models.

The problem of coordination is also interesting and useful. Coordination means requiring certain forms to be processed together. Thus, forms must be allowed to wait in a station for other related forms before they are processed. Coordination is typically modeled with Petri Nets.<sup>6</sup> These models, however, do not deal with performance measures. Combining coordination with performance considerations produces a very difficult problem.

ACKNOWLEDGMENTS

We are indebted to J. Kornatowski for helping to improve this paper.

REFERENCES

1. Baskett, F., Chandy, K., Muntz, R., and Palacios, F., "Open, Closed & Mixed Networks of Queues with Different Classes of Customers," *JACM* 22 (1975), pp. 248-260.
2. Dantzig, G., "Linear Programming & Extensions," Princeton University Press, Princeton, 1963.
3. Ellis, C., "Information Control Nets: A Mathematical Model of Office Information Flow," *ACM Proc. Conf. Simulation, Modeling & Measurement of Computer Systems*, August 1979.
4. Ellis, C. and Nutt, G., "Computer Science & Office Information Systems," Xerox Palo Alto Research Center, June 1979.
5. Ford, Jr., L. and Fulkerson, D., "Flows in Networks," Princeton University Press, Princeton, 1962.
6. Peterson, J., "Petri Nets," *ACM Computing Surveys* 9 (1977), pp. 223-252.
7. "Special Issue on Queuing Network Models of Computer System Performance," *ACM Computing Surveys*, September 1978.
8. Tschritzis, D., "Form Flow Models," Technical Report CSRG-101, University of Toronto, 1979.
9. Zisman, M., "Representation, Specification and Automation of Office Procedures," Ph.D. Thesis, Wharton School, University of Pennsylvania, 1977.

APPENDIX

The FFM can be summarized formally. A form (instance) is a tuple  $\langle \zeta_T, \zeta_K, \zeta_C \rangle$  where

- $\zeta_T \in Z_T$ , the form types;
- $\zeta_K \in Z_K$ , the form keys;
- $\zeta_C \in Z_C$ , the form contents.

An FFM is a tuple  $(S, T, R, A)$  where

- $S = \{s_i\}$  is the set of stations;
- $T = T_I (= \cup_i T_{Ii}) \cup T_O (= \cup_i T_{Oi})$  is the set of in trays and out trays;
- $T_o \subseteq T_I$  is the set of initial trays;
- $T_w \subseteq T_O$  is the set of final trays;
- $R = \{r_x | x \in T_I \& (x \in T_{Ii} \rightarrow \rightarrow (r_x : \{\{\zeta_T, \zeta_K, \zeta_C\} \rightarrow T_{Oi}\})) \cup \{r_y | y \in T_O - T_w \& r_y : \{\{\zeta_T, \zeta_K, \zeta_C\} \rightarrow T_{Ii}\}\}$  is the set of operation and routing selection functions;
- $A = \{a_{xy} | x \in T_I \& y \in \text{range}(r_x) \& a_{xy} : \{\{\zeta_T, \zeta_K, \zeta_C\} \rightarrow \{\{\zeta_T, \zeta_K, \zeta_C\}\}\}$  is the set of operations.

Furthermore, for convenience:

- $E = E_A (= \cup_i E_{Ai}) \cup E_C$  is the set of flow arcs with
- $E_{Ai} = \{(x, y) | x \in T_{Ii} \& y \in \text{range}(r_x)\}$ ,
- $E_C = \{(y, x) | y \in T_O - T_w \& x \in \text{range}(r_y)\}$ .

A number of constraints are required to maintain consistency. For the trays,

$$\begin{aligned} \forall i \neq j, T_{ii} \cap T_{ij} &= \phi, T_{oi} \cap T_{oj} = \phi; \\ \forall i, T_{ii} &\neq \phi; \\ T_{\alpha} &\neq \phi, T_{\omega} \neq \phi. \end{aligned}$$

To ensure all operations and routing are consistent,

$$\begin{aligned} \forall x \in T_I, \forall (\zeta_T, \zeta_K, \zeta_C), y = r_x((\zeta_T, \zeta_K, \zeta_C)) &\rightarrow \langle \zeta_T, \zeta_K, \zeta_C \rangle \\ &\in \text{domain}(a_{xy}); \\ \forall x \in T_I, (y \in \text{range}(r_x) \& y \notin T_{\omega}) &\rightarrow \text{range}(a_{xy}) \subseteq \text{domain}(r_y); \\ \forall y \in T_O - T_{\omega}, \forall (\zeta_T, \zeta_K, \zeta_C), x = r_y((\zeta_T, \zeta_K, \zeta_C)) &\rightarrow \\ &\langle \zeta_T, \zeta_K, \zeta_C \rangle \in \text{domain}(r_x). \end{aligned}$$

It is assumed that if  $\langle \zeta_T, \zeta_K, \zeta_C \rangle$  is input to initial tray  $x$ , then  $\langle \zeta_T, \zeta_K, \zeta_C \rangle \in \text{domain}(r_x)$ . To ensure all trays are connected properly,

$$\begin{aligned} \forall x \in T_I, \text{range}(r_x) &\neq \phi; \\ \forall x \in T_I - T_{\alpha}, \exists y \in T_O - T_{\omega} \& x \in \text{range}(r_y); \\ \forall y \in T_O - T_{\omega}, \text{range}(r_y) &\neq \phi; \\ \forall i, \forall y \in T_{oi}, \exists x \in T_{ii} \& y \in \text{range}(r_x). \end{aligned}$$

In the form flow model, both operation and routing selection functions are allowed. It is possible to derive equivalent models having only the operation or routing selection functions. In these models, each station is split into two parts, one part for operations, the other for routing.

The model having only the operation selection functions (FFM<sub>A</sub>) can be derived from FFM as follows.

$S = \{s^A | s^A \in \text{FFM}(S)\} \cup \{s^C | s^C \in \text{FFM}(S)\}$  is the set of stations with  $s^A$  being the operation part and  $s^C$  being the routing part.

$$\begin{aligned} T &= T_I^A (= \cup_i T_{ii}^A) \cup T_I^C (= \cup_i T_{ii}^C) \\ \cup T_O^A (= \cup_i T_{oi}^A) \cup T_O^C (= \cup_i T_{oi}^C) &\text{ where} \\ T_{ii}^A &= \{x | x \in \text{FFM}(T_{ii})\}; \\ T_{ii}^C &= \{y | y \in \text{FFM}(T_{oi})\}; \\ T_{oi}^A &= \{y | y \in \text{FFM}(T_{oi})\}; \\ T_{oi}^C &= \{e = (y, x) | e \in \text{FFM}(E_C) \& y \in \text{FFM}(T_{oi})\} \\ &\cup \{e = (y, y) | y \in \text{FFM}(T_{\omega})\}. \\ T_{\alpha} &= \{x | x \in T_I^A \& x \in \text{FFM}(T_{\alpha})\}. \\ T_{\omega} &= \{e = (y, y) | e \in T_O^C \& y \in \text{FFM}(T_{\omega})\}. \\ R &= \{r_x | \exists i \& x \in T_{ii}^A \& \text{FFM}(r_x)((\zeta_T, \zeta_K, \zeta_C)) y \rightarrow r_x \\ &((\zeta_T, \zeta_K, \zeta_C)) = y \in T_{oi}^A\} \cup \{r_y | \exists i \& y \in T_{ii}^C \& \\ &(y \in \text{FFM}(T_{\omega}) \rightarrow r_y(*) = (y, y) \equiv e \in T_{oi}^C) \& \\ &(y \in \text{FFM}(T_{\omega}) \rightarrow \text{FFM}(r_y)((\zeta_T, \zeta_K, \zeta_C)) = x \rightarrow r_y \\ &((\zeta_T, \zeta_K, \zeta_C)) = (y, x) \equiv e \in T_{oi}^C)\}. \\ A &= \{a_{xy} | x \in T_I^A \& y \in \text{range}(r_x) \& a_{xy} = \text{FFM}(a_{xy})\} \\ &\cup \{id_{xy} | x \in T_I^C \& y \in \text{range}(r_x)\} \text{ where } id(*) \text{ is the identity} \\ &\text{operation.} \\ E &= E_A^A (= \cup_i E_{Ai}^A) \cup E_A^C (= \cup_i E_{Ai}^C) \cup E_C \text{ where} \\ E_{Ai}^A &= \{(x, y) | x \in T_{ii}^A \& y \in \text{range}(r_x)\}; \\ E_{Ai}^C &= \{(x, y) | x \in T_{ii}^C \& y \in \text{range}(r_x)\}; \\ E_C &= \{(y, y') | \exists i \& y \in T_{oi}^A \& y' \in T_{ii}^C \& \text{FFM}(y) = \text{FFM}(y')\} \\ &\cup \{(e, x) | e = (y, x') \in T_O^C - T_{\omega} \& x \in T_I^A \& \text{FFM}(x) = \text{FFM}(x')\}. \end{aligned}$$

Note that here  $E_C$  must be defined explicitly. The usual consistency constraints apply. The model having only the routing selection functions (FFM<sub>C</sub>) can be derived from FFM as follows.

$$\begin{aligned} S &= \{s^C | s^C \in \text{FFM}(S)\} \cup \{s^A | s^A \in \text{FFM}(S)\}. \\ T &= T_I^C (= \cup_i T_{ii}^C) \cup T_I^A (= \cup_i T_{ii}^A) \\ \cup T_O^C (= \cup_i T_{oi}^C) \cup T_O^A (= \cup_i T_{oi}^A) &\text{ where} \\ T_{ii}^C &= \{x | x \in \text{FFM}(T_{ii})\}; \\ T_{ii}^A &= \{e = (x, y) | e \in \text{FFM}(E_{Ai})\}; \\ T_{oi}^C &= \{x | x \in \text{FFM}(T_{ii})\}; \\ T_{oi}^A &= \{e = (x, y) | e \in \text{FFM}(E_{Ai})\}. \\ T_{\alpha} &= \{x | x \in T_I^C \& x \in \text{FFM}(T_{\alpha})\}. \\ T_{\omega} &= \{e = (x, y) | e \in T_O^A \& y \in \text{FFM}(T_{\omega})\}. \\ R &= \{r_x | \exists i \& x \in T_{oi}^C \& \text{FFM}(r_x)((\zeta_T, \zeta_K, \zeta_C)) \\ &= y \rightarrow r_x((\zeta_T, \zeta_K, \zeta_C)) = (x, y) \\ &\equiv e \in T_{ii}^A\} \cup \{R_e | \exists i \& e \\ &\equiv (x, y) \in T_{oi}^A - T_{\omega} \& \text{FFM}(r_y)((\zeta_T, \zeta_K, \zeta_C)) \\ &= x' \rightarrow r_e((\zeta_T, \zeta_K, \zeta_C)) = x' \in T_I^C\}. \\ A &= \{id_{xx'} | \exists i \& x \in T_{ii}^C \& \text{FFM}(x \in T_{ii}) = \text{FFM}(x' \in T_{ii})\} \\ &\cup \{a_{ee'} | \exists i \& e \in T_{ii}^A \& \text{FFM}(e \in E_{Ai}) \\ &= \text{FFM}(e' \in E_{Ai}) \& a_{ee'} = \text{FFM}(a_{ee'})\}. \\ E &= E_A^C (= \cup_i E_{Ai}^C) \cup E_A^A (= \cup_i E_{Ai}^A) \cup E_C \text{ where} \\ E_{Ai}^C &= \{(x, x') | \exists i \& x \in T_{ii}^C \& x' \in T_{oi}^C \& \text{FFM}(x \in T_{ii}) \\ &= \text{FFM}(x' \in T_{ii})\}; \\ E_{Ai}^A &= \{(e, e') | \exists i \& e \in T_{ii}^A \& e' \in T_{oi}^A \& \text{FFM}(e \in E_{Ai}) \\ &= \text{FFM}(e' \in E_{Ai})\}; \\ E_C &= \{(y, x) | y \in T_O^A \cup (T_O^C - T_{\omega}) \& x \in \text{range}(r_y)\}. \end{aligned}$$

Note here that  $E_A^C$  and  $E_A^A$  must be defined explicitly. The usual consistency constraints apply.

Figure 8 is an FFM<sub>C</sub> derived from Figure 3.

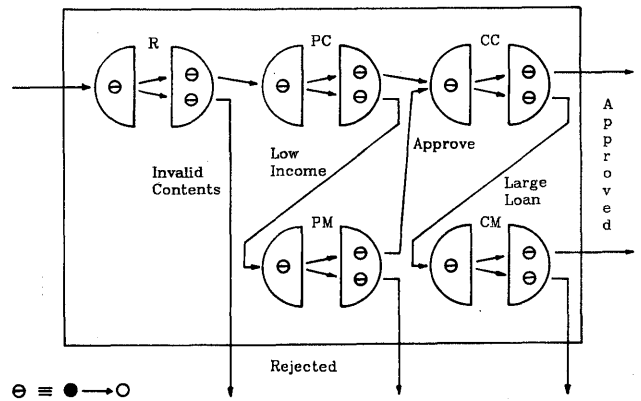


Figure 8—FFM<sub>C</sub> of simpler loan processing office.





# Design principles of an office specification language\*

by MICHAEL HAMMER and JAY S. KUNIN

*Massachusetts Institute of Technology*  
Cambridge, Massachusetts

## INTRODUCTION

Office automation, interpreted most generally, is the utilization of technology to improve the productivity and quality of office work. This concept encompasses a wide range of devices, technologies, tools, and systems. One of its most powerful instances is the notion of an automated office information system. This is a software-intensive, computer-based system that seeks to support (and where appropriate, to automate) an entire office procedure, rather than simply to improve the performance of individual office tasks. However, there is a major impediment to the realization of such systems: because of their application-oriented and office-specific character, they are extremely costly to construct. One of the major reasons for this cost is that office systems analysts lack any tools or methodologies to employ in the process of determining and expressing the requirements of an automated office system. An office specification language is used to describe in a natural yet precise fashion the operation of an office system; its use can improve the process of constructing the system in a number of ways. In this paper, we set forth an approach to the design of office specification languages and present an overview of the major concepts in OSL, one such language that we are developing.

## OFFICE SYSTEM IMPLEMENTATION

Any system or tool that is sufficiently general to be employed without modification in a wide range of office contexts cannot, by definition, be oriented toward the particular needs of a specific office. Such a system addresses a lowest common denominator of office work; thus, its impact on office work and office productivity will inherently be limited. It is only by taking a holistic approach to the activities in an office, by identifying and understanding the office functions performed and the processes conducted to realize them, and by designing and implementing a system to support them, that major improvements in office productivity will be realized.<sup>1</sup> A key concept here is one of an office procedure, an overall framework that provides an organization and order for the individual activities performed in

the office. An automated office information system seeks to improve the execution of office procedures, by improving the performance of specific parts of the procedure and by creating an environment for the integration and control of the procedure as a whole.

An automated office system is an integrated and interconnected collection of components under the supervision of an intelligent control program. These components may be mechanized tools designed to support people in performing unstructured office tasks, or they may be automated subsystems that by means of preprogrammed instructions execute routine and highly structured office tasks. (This distinction between automation and mechanization is fundamental for appreciating the potential of office systems in improving the realization of office work.<sup>2</sup>) By substituting machine labor for human labor where appropriate, and by addressing the entire office rather than just isolated tasks within it, such an office information system represents the paradigm that must be followed to realize the full potential of the new office technology.

However, there is a fundamental problem with this approach to office automation: it is the issue of building office-specific information systems in a cost-effective fashion. No two offices operate in the same way or follow exactly the same procedures; therefore, the paradigm of installing off-the-shelf generic products into an office environment is no longer appropriate when attempting to realize functionally oriented office systems. Instead, a system development effort is required, in which the operations of the office in question are analyzed, its needs assessed, and a custom system designed and implemented for it. The last stage of this process will entail the construction of software that is specific to the particular office in question. This software will embody knowledge of the office's operation; it will automate selected clerical tasks, control the assorted devices employed in the system, and serve as the intelligence that organizes and orders the steps of the office procedure as a whole. This software is clearly specific to the office in question. In other words, custom software must be produced for each office information system.

The difficulty with this approach to office automation is, of course, the fact that it is very expensive to produce office information systems in this way. The process outlined above calls for highly trained personnel (systems analysts and programmers) who must exercise ingenuity in analyzing the

\* This research was supported in part by Exxon Enterprises, Inc.

operation of the office in question, defining its needs, and designing and then implementing a system. Moreover, experience has repeatedly shown that complex software systems produced by conventional means tend to be error-prone, costly to construct, and difficult to change. If we are to be successful in building and installing custom office systems on a wide scale, we must seek new means to produce them.

Let us look more closely at the process of office system construction, and at the problems with it. It is possible to identify four stages in this activity: analysis, specification, design, and implementation. In the first stage, the current operations of the office are studied and their shortcomings identified, and the general capabilities of the automated system to be built are defined; in the next, precise specifications of this system are produced. These are the tasks of the office systems analyst. The programmer then designs the structure of a system that will meet these specifications and finally reduces them to code. The major sources of difficulty in this process lie with the analyst's activities rather than the programmer's. The programmer need only to seek to implement a system that meets the specifications given him; the analyst has the responsibility of constructing these specifications. His is a challenging and creative job; yet the analyst lacks any useful methodologies or tools to employ in analyzing an office or specifying a system for it. His task as a whole lacks structure; there are few guidelines or principles for him to employ.

One particular problem the analyst faces is that he has no effective notation or language in which to express himself. Many errors in software systems arise from the fact that the original specifications for the system are unclear, incorrect, or incomplete; this derives from the fact that they are poorly expressed in a language unsuitable for the purpose. Currently, an office analyst will use English to describe the current operation of an office as well as to specify the desired functionality of an automated system that is to be built. Although rich and expressive, English, like all natural languages, is imprecise and ambiguous and consequently not useful for the accurate specification of systems. Specifications must bridge the gap between the analyst (who is oriented toward application constructs and office needs) and the programmer (who is concerned with the design and implementation of software systems). This gap is the breeding ground for ineffective communications, expense, and error.

## OFFICE SPECIFICATION LANGUAGES

We believe that many of the problems discussed above can be significantly mitigated by providing the analyst with a problem-oriented office specification language. This is a formal language for describing in high-level and machine-independent terms the operation of an office system (either manual or automated). It may be thought of as a notation in terms of which an office system analyst can express himself, both for describing an existing office operation and for specifying the operation of an automated system to be built

by the programmer. A specification language is a formal language, with rigorously defined syntax and semantics. Thus, any description expressed in it is unambiguous and open to a single interpretation. Furthermore, the primitives of a high-level language are based on the natural structures and vocabulary of office work so that the language user can express himself in terms natural to the problem domain. Such a language can serve as an effective means for specifying in a precise, natural, and understandable way the operation of an office system.

We envision a variety of potential uses for such a language. The principal one is as a communications mechanism between office systems analyst and programmer. Because of its formality and precision, specifications expressed in the language can be clearly understood and interpreted by the programmer who must use them as the basis for his system implementation effort. The use of this language will enable an office systems analyst to describe more precisely to a programmer the system that is to be constructed; this improved communication can have a major and positive impact on the systems thus produced, improving their quality and lowering their cost. The use of such a language facilitates the jobs of both the analyst and the programmer. Because of the high level of the language, the analyst will be able to readily express himself in terms familiar to him while suppressing irrelevant detail. Second, the language can impose a structure on the entire process of office analysis and system specification. By providing the analyst with high-level primitives in terms of which he is to express a system, a specification language effectively gives him a set of templates with which he is encouraged to analyze office operations. Thus, the analyst is presented not just with a set of disconnected language features but with an approach to their employment, a perspective on office operation that provides a conceptual framework in terms of which to analyze and describe office operations. Finally, there are several uses of such a language that are not directly related to the process of constructing automated office systems. A formal specification language for office procedures can serve as a very effective mechanism for expressing precise and complete descriptions of the existing manual office operation. In current practice, English is the language employed in systems and procedures manuals; however, as is well-known, these manuals are usually incomplete, difficult to read, and obsolete. Well organized and precise specifications in a high-level language can be used as a reference for office workers in many office environments. Related uses are for the training of new employees, and for the recording of organizational history in a way that survives the coming and going of individual office personnel. The formal specifications of an office procedure can also be subjected to various analytic techniques in an effort to identify bottlenecks and problem areas in its operation; this can highlight those areas of the procedure most in need of rationalization and redesign, whether or not in the context of an automation effort.

Obviously, the mode in which an office specification language is used depends in part on the application for which it is being employed. However, in general, we expect that

specifications will be written by a trained office systems analyst who has been instructed in the use of the language. This person will not necessarily be a computer expert; he may be a manager, a staff professional, a secretarial or clerical worker, or a specialist dedicated to this task. He must possess two important skills: a deep understanding of office work, and an ability to analyze and describe office operation in a systematic fashion.

We believe that the use of the language will be ongoing but intermittent. That is, at some point the initial description of an existing or proposed office system will be expressed in the language, and on a regular basis this description will be updated to reflect changing circumstances and evolving needs. However, we do not believe that specifications will be modified on an *ad hoc* basis by individual office workers. While there will be few writers of specifications, we expect there to be a large population of people who will want to read specifications expressed in the language. Readers of office specifications will include office workers who will consult them in order to determine aspects of procedures with which they are not familiar; office trainees in the process of learning the office operation; office managers seeking ways in which the operations of the office can be improved; and programmers who will be called upon to translate such specifications into operating programs.

As suggested above, we believe that this language will be used both for prescriptive and descriptive purposes; that is, to describe an existing office operation as well as a new and proposed one. In fact, such uses are often demanded in the context of an evolving office system. An analyst must first construct a description of the system as it is currently configured and use that as a basis for developing specifications of a new and improved system. It is rarely feasible to institute a revolutionary change in the process of an automation effort and to dramatically restructure an entire office operation; rather, the new office system must evolve from the old one. Consequently, at some suitable level of abstraction, the specifications of the new system should be virtually identical to those of the old one. It is only at the level of mechanism and implementation that the two become distinguishable. Thus, it is appropriate that the specification language be multi-tiered, with the topmost level expressing the implementation-independent structure of the office and only the more detailed level serving to identify the particular way in which the general structure is being instantiated.

#### AN APPROACH TO SPECIFICATION LANGUAGE DESIGN

We are engaged in an ongoing effort to develop such an office specification language (known as OSL) for use in analyzing, describing, and implementing office systems. Based on the foregoing perspective on the use and utility of such a language we have developed the following approach and design criteria that we are employing in this effort.

1. The language must be formal and well defined; that is,

it will have a limited vocabulary of constructs that can be combined only in specific ways. As a result, it will be possible to determine in an automatic fashion whether a particular specification in the language is legal and meaningful. Furthermore, any legal specification will admit of only one interpretation. These properties of formally defined languages avoid many of the difficulties associated with English and other natural languages.

2. The language must be highly readable; it should be possible for an individual with a very small amount of training to be able to read and understand specifications expressed in OSL. This criterion is motivated by the fact that there will be a large readership for OSL specifications, most of whom will not be specialists in the language. There are several consequences of this requirement. First, it dictates that the constructs of OSL be natural and problem-oriented rather than general and abstract. That is, the dictions of the language should reflect the natural semantics of offices and office work; the language primitives should directly correspond to office activities and structures so that the description of an office procedure will be couched in terms meaningful to those familiar with office work. Furthermore, every specification expressed in the language should have a manifest and understandable overall organization and structure. That is, not only should individual atoms of the language be easy to comprehend, but a description of an office procedure as a whole should be organized in a way that enables people to comprehend it easily.
3. The language should support the process of writing descriptions by incorporating a standard and natural logical structure for office specifications. We believe that the way to aid someone seeking to write specifications in a language is not by providing him with a minimal set of general and flexible linguistic features. While it may be easy to learn the meaning and capability of each one of these constructs, the entire burden of combining them into a complete specification is then thrust upon the user. A language that is easy to learn is often difficult to use for all but the most trivial of applications. Consequently, we think it appropriate that OSL possess a rather more complex and intricate inherent structure, which may require more effort to learn but which should greatly enhance its usability. In other words, the user of OSL will start out with a preconceived notion of the general structure of the specifications he will produce; his task is to match the particulars of the office in question to the canonical structure. As a consequence, associated with OSL will be a methodology for conducting office analyses and writing specifications, which is based on the same conceptualization of office work embedded in the language.
4. The language should be high-level and nonprocedural. The specification of an office procedure in OSL will be expressed at a level of abstraction corresponding to

the functionality and purpose of the procedure, rather than in terms of the low level task structure used to implement it. The focus will be on what the procedure does, rather than on the details of how it does it.

5. Specifications expressed in the language must be modifiable. Office procedures are highly dynamic; they continually evolve to meet unanticipated situations and new requirements. The specifications for an office system must consequently evolve in an incremental fashion to reflect these new developments; if they are not readily modifiable, they will inevitably become obsolete and unused. Moreover, modular and modifiable specifications can result in more maintainable software systems. If the software system reflects the structure of the specifications, then as the specifications change, the implementation can often be modified in corresponding and limited ways.

The overall goal of our design effort is to develop a language in which office systems analysts can readily construct highly readable specifications that are clear, unambiguous and natural descriptions of office procedures. These specifications should uncover and highlight the basic structure of the procedure rather than focus on the details involved in its implementation. Our approach is a functional one; that is, we do not find it feasible or even desirable to attempt to capture in a specification all of the mechanisms associated with an office procedure. First, any such "complete" specification will be overwhelming in its size and complexity. Second, it is unlikely that an office system analyst (in any finite amount of time) will be able to uncover all possible variations of the procedure. And, third, the implementation details of an office procedure continually evolve as office workers develop new techniques to solve old problems or face previously unencountered difficulties. Consequently, we have not sought to achieve any elusive "completeness." Instead, a description couched in OSL will focus on the purpose of the procedure, rather than on its mechanics. This is accomplished by including in the language primitives that express the goals of office activities in application terms. In order to achieve this end, it is necessary to sacrifice completeness in another way as well. We do not expect that OSL will be appropriate for describing all conceivable office procedures. OSL embodies a particular perspective and approach to office work and its description which, we believe, matches a large number of office procedures, although certainly not all of them. Our goal is to make OSL extremely usable for a large class of applications, rather than minimally adequate for the universal class of applications. We have sought to optimize the design so that what OSL does, it does very well; we are devotees of the 80/20 rule. Whether or not we achieve this goal and whether or not the class of applications for which OSL is appropriate will be large enough to justify this decision, only extensive experience with the language will answer.

The fundamental premise underlying the design of OSL is that there is a high degree of commonality of structure and activity among procedures in different offices. In other

words, we believe that there are fundamental semantic structures in the office application domain that recur in many different contexts. This commonality can be exploited by identifying the structures that are repeatedly used in natural descriptions of different office procedures and embedding them in a formal language. The user of the language will then find that it provides him with just those problem-oriented features that he wishes to use; he will not have to build up a description of an office procedure from lower-level and more general constructs. Consequently, the design of an office specification language must be based on an extensive familiarity with the application environment.

The first (and an ongoing) aspect of the OSL design effort has been to conduct many case studies of office procedures in different environments. We have conducted analyses of a large number of operational and administrative offices, of several different kinds, within a variety of organizations. It is by analyzing and abstracting from these descriptions that we have identified the fundamental constructs of OSL. Based on these analyses, we have concluded that most office procedures are fundamentally simple processes that are often obscured by implementation details and disorganized exception handling. However, when it is eventually uncovered, the basic structure of the office procedure is often relatively easy to comprehend and describe, given the appropriate set of primitives. The "complexity" of office procedures is often an artifact; the goal of OSL is to manage and even avoid this complexity.

The following summarizes the principal characteristics of OSL.

1. A specification expressed in OSL takes a holistic view of office activities; it is not based on a description of the processing performed on individual forms passing through an office nor around the activities of individual office workers. Rather, it expresses an integrated view of the office activities as a whole, with the focus on the end being achieved rather than the means being employed to achieve it.
2. OSL descriptions are expressed in terms of application-oriented constructs, eliminating as far as possible any detail that is not germane to the application itself but that results only from the fact that the specification is being expressed in a formal language.
3. OSL specifications are highly structured in a canonical way. OSL imposes a uniform format on the description of every office procedure. Furthermore, this description is modular, so that it is possible to develop an understanding of individual parts of it and to comprehend its overall structure without working through all the details. This modularity is accomplished in two ways. First, the language employs techniques of successive refinement so that the procedure can be expressed and understood at multiple levels of abstraction. Second, because of their importance in office procedures, the descriptions of exceptions and special cases are not incorporated directly into the main line

of the procedure, but are attached to it in specific and predetermined ways.

4. OSL embodies the most common specialized constructs used to describe office procedures. While this does lead to growth in the number of language features, with some attendant increase in its complexity, it also leads to shorter and more understandable specifications. The intent of the procedure is evident from its surface, since it is being expressed directly rather than coded in terms of general and abstract facilities.
5. OSL makes extensive use of declarative specification techniques. That is, as much knowledge of the procedure and its operation as possible is embedded not in a description of activities to be performed, but as constraints and restrictions on the data values or documents associated with the procedure. This leads to greatly simplified procedural descriptions as well as specifications that are easier to change (because of the locality of this information).

Below we shall see how these general principles and criteria have been addressed by the current version of the language.

## AN OVERVIEW OF OSL

The major premises discussed above have had a major influence on the development of OSL and are reflected in its philosophy and features. A holistic view of office specification is central to the structure of the language. While OSL recognizes the importance of forms and people as individual units of office activity, it does not structure a procedure description around them; its orientation is toward the *objects* in an office. Objects in this context are the entities that are the focus of office activities and that form the basis for a description of office functions; the office as a whole is described in terms of the evolving history of its objects. OSL provides canonical high-level office-oriented constructs for the specification of both data and control structures. Such constructs provide a framework for the organization and presentation of a specification and also act as a guide to the analyst in structuring his task. This framework in turn provides for the readability and naturalness of expression necessary for using OSL in documentation and training. Finally, OSL provides a built-in structure for all specifications; the office description as a whole has a standard format, and each of its components can be decomposed in a uniform way.

We shall now describe in some detail the structure of the initial version of OSL. In this discussion, we will employ as an example the Office of Sponsored Programs (OSP) at MIT, whose major functions are to expedite the submission of research proposals and the negotiation of contracts, and to monitor the resulting grants and contracts to ensure compliance with internal policies and contractual requirements.

An office specification in OSL consists of two major components: a description of the application domain with which

the office is concerned, and specifications of the procedures performed in the office. The former provides a context for the description of the procedures. It effectively expresses a model of the world of the office; it describes the objects on which the office is focused, the organizational context of the office, the documents and forms that the office processes, and the information that it needs to utilize. In the case of the OSP, this contextual information describes a world consisting of proposals, contracts, funding agencies, researchers, laboratory directors, and the like. The description is couched in terms of a variant of the SDM,<sup>3</sup> a data modeling mechanism originally developed for describing the information content of databases. The key feature of the SDM is that it models an application rather than data; thus the specification includes a direct description of the office and its environment. This enables the specification to distinguish substance from artifact; a procedure can access information it requires by directly referring to the appropriate attribute of an entity, without caring whether that information is captured on a form, in a database, or elsewhere. This "schema" thus naturally expresses the static semantics of the office in terms with which a reader is likely to be familiar.

The description of the office environment is expressed in terms of entities and their attributes, inter-entity relationships, and entity collections. Associated with the description of an office entity is the definition of those documents related to it (for example, a proposal document is associated with each proposal entity), as well as constraints on the attributes of the entity and on its processing (for example, that the principal investigator of a contract must be a faculty member or that if human subjects are to be used in the research then approval must be obtained from an appropriate university committee). By associating constraints with objects and documents, the description of context becomes more meaningful and the specification of the procedures becomes simpler and more modular. Specialized entity and relationship types (such as people, agreements, schedules, logs, supervision, and the like) are provided, since they recur frequently and they possess special semantics. Included in this environmental specification is a description of the offices in the organization and the lines of communication and authority that connect them. Such an organizational description is particularly valuable when a function is realized by means of related procedures executed in different offices. The local office context describes the people in the office, their roles, responsibilities and authority, as well as the files maintained in the office. The environmental description also identifies the primary objects of the office. These are the entities that are the major focus of the office activities and around which the descriptions of the procedures are organized. In the OSP, the primary objects are proposals and contracts.

The dynamics of the office are captured in the specification of its procedures, the activities it performs in the described context. Just as an appropriately-designed data model can serve as the basis for natural descriptions of the environment, so too a simple but powerful model of office activities can be applied to procedural specification. To this

end, OSL incorporates a canonical set of structures for process description that are based on three concepts: an orientation around objects; hierarchically structured and modular descriptions; and an emphasis on the identification of exceptions.

Fundamentally, every OSL procedure is concerned with processing and/or managing a primary object. Based upon such an object orientation, we find that a large number of office activities can be described in terms of a fairly simple three-stage model representing the "life cycle" of an object: an *initiating* procedure, an *administrative* procedure and a *terminating* procedure. The primary object has three versions, which correspond to the three stages of the procedure.

An initiating procedure manages an *initiator* object. This is an "active" procedure; it "pushes" the initiator through a series of operations to an end point, at which time the administrative procedure is triggered. In the OSP, the initiator is a contract proposal, and the initiating procedure is concerned with obtaining institutional approval for the proposal and negotiation with the proposed sponsor. At the conclusion of the initiating procedure (which may include several iterations of the proposal submission process), the proposal is either rejected, terminating the overall procedure, or accepted, invoking an administrative procedure.

An administrative procedure manages a (set of) *resource* objects. It is concerned with maintaining the status of the resource and verifying and recording all activities that are applied to it. In the OSP, the resource is a contract; the process involves assuring that the spending restrictions and reporting requirements of the contract are honored.

The final stage of the "life cycle" of an object is termination. When the resource is no longer of interest, the administrative process triggers a termination procedure, which produces an *archive* object. This is another "active" procedure, although in many cases it is relatively uncomplicated. An archive object simply represents any information that must be available after the termination of the procedure and destruction of the resource.

We have thus defined two classes of procedures: administrative and active. An administrative procedure is specified by means of a formatted description that identifies periodic inputs and outputs and how they are to be handled (progress reports and accounting reports in OSP) and nonperiodic, but expected, events, together with the processing required to handle them (e.g., purchase authorizations).

Active procedures are specified somewhat differently. An active procedure can be viewed as the application of a set of activities ("verbs") to an object by a responsible agent ("subject"). The control structure for specifying the order in which activities are to be applied is formalized in the following manner:

The procedure is described in terms of states, events, and activities. States are stages in the execution of the procedure at which no further processing can be done until the occurrence of some event. An event is an autonomous occurrence that is beyond the control of the agent responsible for the procedure (e.g., the receipt of a document, the arrival of a specific date and time). When the procedure is in a given state, it is waiting for the occurrence of one of a designated

set of events; when one of these events occurs, a corresponding activity is executed, at the conclusion of which the procedure is left in some other state. A state machine-like formalism can be used to express these relationships and determine the overall control structure of the procedure. For example, one state of the OSP initiating procedure corresponds to a situation in which a proposal has been sent to the legal office for review; when an appropriate event (receipt of a response from the legal office) occurs, the proposal can be further processed by OSP preparatory to its submission to the proposed sponsor.

Activities are the specific actions performed in the course of a procedure; the description of an activity, like that of a procedure as a whole, is hierarchical and modular. At the top level of specification, activities are specified with a minimum of detail; this is provided in the lower levels of description. This hierarchical structure supports the modification of activity descriptions; it also allows for both descriptive and prescriptive (normative) specifications of any given office situation. We anticipate that the top-level specification will express the goal of the activity; the detail may represent either the results of the analysis (description) or the specification of an implementation (prescription).

At the base of the activity hierarchy is a set of activity *primitives*, each of which is a fundamental office operation. For example, the primitive "select" is used to describe a decision in which a subset of available resources will be chosen. (The "allocate" primitive indicates the opposite situation.) Note that even at the primitive level, we seek to describe function, rather than implementation. Thus, "select" may be performed in various ways; we may have an algorithm for implementing it (e.g., to select the highest-scoring applicant), or it may be inherently judgmental (e.g., to select a site for a new plant). This flexibility in describing activities is desirable in a tool for analysis and documentation.

One of the key aspects of our formulation is its approach to exception handling. As we have discussed, special cases and exceptions are often the source of the perceived complexity of many office procedures. By organizing the description of exceptions, we provide a means of making the overall specification more organized and readable.

We take a hierarchical approach to the specification of exceptions. Each level of procedure and activity description has an associated list of exceptions. These are classified in terms of the nature of the events that give rise to them; instances include violation of timing constraints, invalid data values, unavailable personnel, and activity-specific errors (e.g., an inadequate set of resources from which to make a stipulated selection). The responses to the exceptions are also frequently drawn from a canonical set. The descriptions of the exceptional situations and the responses to them are separated from the mainline procedure to enhance the latter's readability.

## RELATED WORK

We note that our usage of the concept of a specification language differs from that typically employed in the computer science literature.<sup>4</sup> The common usage refers to a gen-

eral-purpose facility for describing the behavior of individual modules of an arbitrary large software system. By contrast, our perspective is domain-specific and implementation-independent. However, some earlier work has been done in areas related to our effort, generally in the context of business information system design. The Time Automated Grid<sup>5</sup> and Accurately Defined Systems<sup>6</sup> languages are designed for describing file processing applications; the latter is primarily a documentation tool. The Business Definition Language<sup>7</sup> is also aimed at highly-structured data processing tasks. The Problem Specification Language<sup>8</sup> is a more general language for defining information system requirements. Although a number of ideas useful for the specification of office systems can be derived from these languages, it is clear that their scope is inadequate for the flexible, interactive, and semi-structured nature of the systems that we are addressing.

Recently, several attempts have been made to design languages specifically for the office domain. Barber and Hewitt<sup>9</sup> are using a form of the Actor formalism<sup>10</sup> to specify the activities of office workers and the communications among them, primarily in an effort to find means of symbolically proving, simulating and modifying office procedures. IBM's System for Business Automation Programming Language<sup>11</sup> is based upon the Query-by-example relational database query language.<sup>12</sup> The user programs in a forms-oriented graphical environment; primitives in the language include database access, forms editing and control, and similar functions. The Xerox PARC Officetalk system<sup>13</sup> uses a similar forms-oriented interface and programming-by-example paradigm, although it also allows the writing of procedural descriptions of forms processing with a small set of filing, communications, and forms editing primitives.

A major problem with all these languages is that they do not deal directly with office function. They are oriented toward the tasks of individual office workers; the focus is on current task structure, rather than fundamental function. As a result, the utility of these languages for high-level specification is limited.

The Office Procedure Specification Language<sup>2</sup> does deal more directly with office functions. It primarily allows the description of documents and communications patterns. Primitives are at the level of document definition and movement; more complex processing is expressed by programs written in a general-purpose programming language. The structure and syntax of the language are tightly coupled to an augmented Petri Net formalism, a general representation scheme for asynchronous, concurrent processes. The major shortcoming of this approach, which it shares with all of the above office description languages, is that it lacks any constructs at a level higher than "send message" or "file document." Nor do any of these languages result in highly structured or readable specifications.

## SUMMARY

We have presented the basic concepts of an office specification language and identified the principles on which the design of the OSL design effort is continuing. A first version of the language has been specified, and it is being applied to a number of test cases. Experience with the use of the language will indicate needed changes; we expect that this iterative approach will lead to a highly effective language. We have also begun exploring the design of an OSL processing system, which would seek to automatically generate an office information system from its OSL description.

## ACKNOWLEDGMENTS

We would like to acknowledge the valuable suggestions of our colleagues Marvin Sirbu and Sandor Schoichet.

## REFERENCES

1. Hammer, M. and Zisman, M., "Design and Implementation of Office Information Systems," *Proc. NYU Symposium on Automated Office Systems*, New York University Graduate School of Business Administration, May 1979, pp. 13-24.
2. Zisman, M. D., *Representation, Specification and Automation of Office Procedures*, Ph.D. Dissertation, The Wharton School, University of Pennsylvania, 1977.
3. McLeod, D. J., *A Semantic Data Model and Its Associated User Interface*, Ph.D. Dissertation, MIT, Department of Electrical Engineering & Computer Science, August 1978.
4. Liskov, B. and Berzins, V., "An Appraisal of Program Specifications," T. Wegner (ed.), *Research Directions in Software Technology*, MIT Press, 1979.
5. IBM, "The Time Automated Grid System (TAG): Sales and Systems Guide," Publication GY20-0358-1, IBM, May 1971.
6. Lynch, H. J., "ADS: A Technique in System Documentation," *Database*, Vol. 1, No. 1, Spring 1969.
7. Hammer, Michael, Howe, W. Gerry, Kruskal, Vincent J., and Wladawsky, Irving, "A Very High Level Programming Language for Data Processing Applications," *CACM*, Vol. 20, No. 11, November 1977.
8. Teichroew, D., "Problem Statement Analysis: Requirements for the PSA," J. D. Couger and R. W. Knapp (eds.), *System Analysis Techniques*, John Wiley and Sons, 1974.
9. Barber, G. and Hewitt, C., "Towards the Development of Office mantics," Draft, October 1979.
10. Hewitt, C., "Viewing Control Structures as Patterns of Passing Messages," Memo 410, MIT Artificial Intelligence Laboratory, December 1976.
11. Zloof, M. M. and de Jong, S. P., "The System for Business Automation (SBA): Programming Language," *CACM*, Vol. 20, No. 6, June 1977.
12. Zloof, M. M., "Query-by-example," *Proc. NCC, AFIPS*, 1975.
13. Newman, W., "Studies of Office Procedures and Information Flow," Internal Memo, Office Research Group, Xerox Palo Alto Research Center, May 1976.





# Automated workflow control: A key to office productivity

by L. S. BAUMANN and R. D. COOP

*Electronic Office Research Project, Sperry Univac  
Roseville, Minnesota*

## INTRODUCTION

Until the mid nineteenth century, labor in the United States and the rest of the world was primarily manual in nature. Productivity improvements were obtained by improving workflow or decreasing wages. After the start of the industrial revolution, a third variable affecting productivity arose. Mechanization (later to be called automation with the introduction of the computer) allowed product output per worker to be greatly increased. Mechanization was applied to those work tasks which were well structured and easily implementable. These areas were primarily the farm and factory manual labor activities which were very expensive in terms of manpower. Production increased sharply and continues to make steady improvement even today.

One class of worker (the so-called white collar worker) was left out of this revolution. Because of the basic lack of structure in white collar work, mechanization was limited primarily to clerical and secretary-typist areas. However, even with this lack of mechanization the productivity of the white collar workforce kept pace with factory and farm productivity until the mid twentieth century. By that time, automation had developed to such an extent in factory operations that a divergence began to appear between factory and white collar productivity. For example, during the period from 1965 to 1975, factory productivity [1] in the U.S. electronics industry increased by 3.6 percent per year compared to 2.9 percent per year for the white collar workforce. Obviously, the overall worker productivity was retarded by the lag in white collar productivity.

Recent studies [2,5,6,7,8,11] indicate that productivity in the office is based on the utilization of principals (managers and professionals). Productivity logs are due primarily to the elimination of support to principals over the last several years and to lower work expectations. Looking at the productivity log we find that the time lost in work interchange is the chief non-production cost. An attack on work interchange, management control of work, and support responsiveness can pay considerable dividends in improved office productivity. To begin this attack, we must first get a better understanding of the nature of office work. Our second step will be to create a mathematical model of a general office process which clearly shows workflow control as an isolatable and mechanizable process. On the basis of this devel-

opment of workflow control within the model, we briefly describe the elements of workflow control. Mechanization of these elements will improve office productivity.

## THE NATURE OF OFFICE WORK

What constitutes a business office? This question is easily answered at the macro level. An office [2,10] consists of people interacting in an environment to carry out the mission of a business. The environment provides the resources used to aid the interaction among people. The human interface (to environment or through environment to other workers) fully characterizes the work at each step in office processes. The workings of the interface and the worker's psychological relationship to the interface define discrete stages in the work process.

The concept of "process" is not as clear as it seems. A process is an on-going activity in which data or materials flow in and out. A snapshot in time shows each datum or material element in a different stage of process completion. If we break a major process or activity into individual subprocesses, we find in a snapshot that different combinations of these individual subprocesses are operating on different datum or material elements.

If we examine an office organization, we find a hierarchy of management personnel controlling the office workers. Slicing into this hierarchy at any point, we find a manager (at an appropriate hierarchical rank) and working personnel (who also may be management personnel or office workers). The manager has an office mission which is carried out by making work assignments to the working personnel. The working personnel report on the status of the assigned work to the manager providing feedback for his evaluation and decision making. The work assignment is a high level process, hereafter called an *office process*. Office process definitions vary from office to office, and from point to point in the organizational hierarchy.

An *office process* operates on information over time. At any instant we find working personnel carrying out concurrent single worker processes. Moving the snapshot in time introduces the sequentiality of worker activity, which gives us the basis for creating our model. This segmentation process is carried out on the office process to isolate each in-

dividual single worker process (hereafter called a *personal process*).

Figure 1 part A shows an office process partitioned into  $m$  time slices. Each time slice is a concurrent multiworker work profile which can be partitioned into concurrent processes for each worker, as shown in part B. Part C shows the concurrent single worker processes partitioned into unique personal processes. The personal process may be carried out at a manual or electronic office station, as shown in part D. The worker can perform his personal processes at either a manual office station or a partially mechanized electronic office station. In the latter case, each personal process must be separated into manual and mechanized subprocesses.

The ability to perform any personal process manually or with mechanization is very important in an automated office. Loss of processed information is intolerable. Since any electronic aid can fail, the process must be able to continue manually. Thus the manual and electronic personal process models must be similar, and allow easy transition from one to the other at any point in the process. Recovery actions

required in this transition are non-productive and therefore must be minimized.

Personal processes consist of subprocess transformations and information/topological states. Manual office activity (see upper part of Figure 1 part D) involves a *workspace*, *files*, and *worker* topology and subprocess transformations carried out by the worker. When we begin to mechanize the topology expands (see lower part of Figure 1 part D) and we must break up the process transformations. The manual subprocess involves a *manual workspace*, *manual files*, and *worker* topology with subprocess transformations carried out by the worker. This is very similar to manual office activity. In addition, mechanization requires a mechanized subprocess involving a *mechanized workspace*, *mechanized files*, and *Electronic Data Processing (EDP)* topology with subprocess transformations carried out by EDP.

The transformation of information in a personal process can have two forms: *structured* and *unstructured* [12,13].

A structured transform is a process transformation which is computable within the processing and storage resources of the available Electronic Data Processing (EDP). Unstruc-

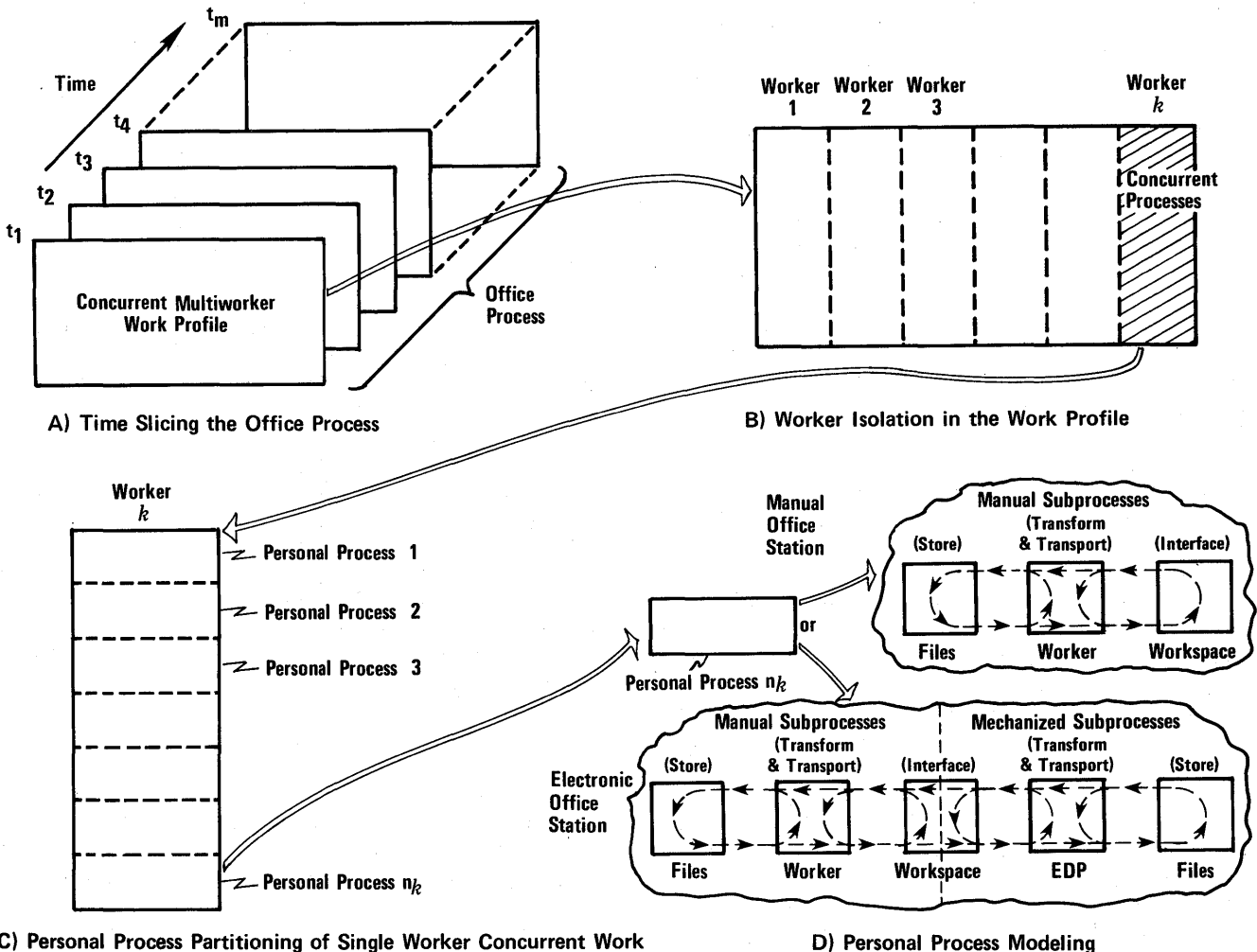


Figure 1—Segmentation and modeling of the office process.

tured transforms are those which are not computable by the available EDP technology and must remain relegated to the worker. The workspace acts as the interface between these two classes of transformations and is the topological location of the resultant process transformation. An interesting sidelight is the topological division of files into manual and EDP classes because of the transformation division. The mechanized office is a true distributed processing system even if the EDP is centralized.

**MODELING OFFICE WORK**

Our next objective is to construct a model of office work which defines the concept of workflow control. To accomplish this end we will return to Figure 1 and create a mathematical formulation. Zisman [4] has shown that office processes have a knowledge domain and a process (action) domain. The knowledge domain controls the activity in the process domain. A model of a process can be considered a computable ordered graph  $P[Q \times C]$  where

- $P$  is a Petri Net [14],
- $Q$  is the knowledge domain state set,
- $C$  is the process domain state set, and
- $Q \times C$  is the cartesian product.

We can begin the creation of a model for office work by examining Figure 1 part D. We can model the *manual office personal process* as:

$$P_i[Q \times C] \text{ for } i=1,2,\dots,k \text{ workers}$$

An *electronic office personal process* is broken into two subprocesses; a manual subprocess model:

$$P_j'[Q \times C] \text{ for } j=1,2,\dots,k' \text{ workers}$$

and a mechanized model:

$$P_j''[Q \times C] \text{ for } j=1,2,\dots,k' \text{ workers}$$

Looking at Figure 1 part C, we can see that concurrent personal processes are possible for each worker's timeslice. This concurrency can be represented as the union over  $n'$  electronic office personal processes and  $n$  manual personal processes:

$$U_{\alpha=1}^{n_j'} \left\{ \begin{matrix} P_{j\alpha}' \\ P_{j\alpha}'' \end{matrix} \right\} \cup U_{\beta=1}^{n_j} \{P_{i\beta}\}$$

Moving to Figure 1 part B, we see that a work profile consists of concurrent multiworker activity which may be represented as the union over all workers:

$$U_{j=1}^k U_{\alpha=1}^{n_j'} \left\{ \begin{matrix} P_{j\alpha}' \\ P_{j\alpha}'' \end{matrix} \right\} \cup U_{i=1}^k U_{\beta=1}^{n_i} \{P_{i\beta}\}$$

In Figure 1 part A, the work profiles are sequenced over  $m$  time slices. We will represent this sequencing as operation  $S$  which selects the appropriate work profile models. This

operation may be represented as:

Office Process

$$= S_{\gamma=1}^m \left[ U_{j=1}^{k_j} U_{\alpha=1}^{n_{j\alpha}'} \left\{ \begin{matrix} P_{j\alpha}' \\ P_{j\alpha}'' \end{matrix} \right\} \cup U_{i=1}^{k_i} U_{\beta=1}^{n_{i\beta}} \{P_{i\beta}\} \right]$$

This formulation is greatly simplified if we use the properties of Petri Nets and recognize that the concurrency/sequentiality of the personal processes can be formulated by a model in the knowledge domain guide by process domain production rules (see Zisman [4]).

The office process formulation becomes a set of graphs:

Office Process

$$= \left\{ \begin{matrix} P_c[Q] \\ P_{ir}[Q \times C] \\ P_{js}'[Q \times C] \\ P_{jt}''[Q \times C] \end{matrix} \right\} \text{ for } \begin{matrix} i=1,2,\dots,k \\ j=1,2,\dots,k' \\ r=1,2,\dots,(n_1+n_2+\dots+n_m) \\ s=1,2,\dots,(n_1'+n_2'+\dots+n_m') \\ t=1,2,\dots,(n_1+n_2+\dots+n_m) \end{matrix}$$

where  $P_c$  is the sequentiality/concurrency knowledge model.  $P_c$  can be further subdivided into a graph  $P_c'$  representing the office manager knowledge operation and  $P_c''$  representing the actual control of workflow. This yields our final model:

$$\text{Office Process} = \left\{ \begin{matrix} P_c' \\ P_c'' \\ P_{ir} \\ P_{js}' \\ P_{jt}'' \end{matrix} \right\} \text{ for all } i, j, r, s, \text{ and } t.$$

We assume that the human worker is a consolidated graph ( $P_j$  and  $P_i$ ) for all  $r$  and  $s$ . We also assume that  $I$  unique mechanized subprocesses ( $P_{\sigma}''$ ,  $\sigma=1,2,\dots,I$ ) make up the  $P_{jt}''$  concurrent subprocesses. Figure 2 shows the final office process model with a manager  $P_c'$ , multiple manual workers ( $P_i$ ,  $i=1,2,\dots,k$ ), multiple mechanized workers ( $P_j'$ ,  $j=1,2,\dots,k'$ ), multiple mechanized subprocesses ( $P_{\sigma}''$ ,  $\sigma=1,2,\dots,I$ ), and a workflow control system ( $P_c''$ ).

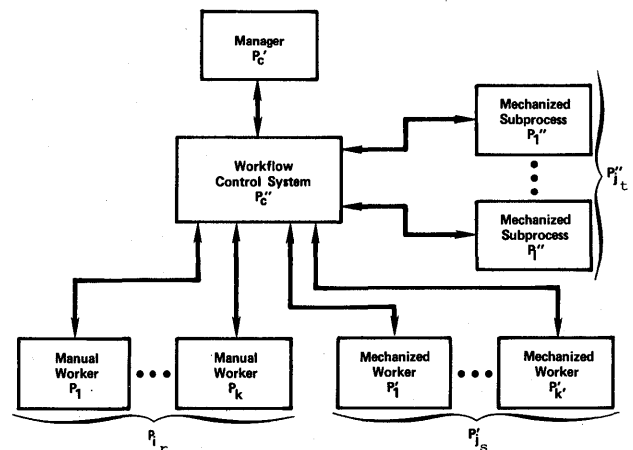


Figure 2—Physical representation of office process model.

THE ELEMENTS OF WORKFLOW CONTROL

On the basis of the previous development of workflow control (which demonstrates existence and isolatability) within the model, we now examine the basic concepts and elements of workflow control to demonstrate its mechanizability. Consider an office as a room containing people and facilities (mechanisms, desks, chairs, pencils, paper, files, lighting, heat, etc.). What is characteristic of this office *without regard* to specific work? The office is both the control tower and the service station for a business. By assembling and analyzing information, it exercises control over distribution, production, procurement, and finance. By assuming responsibility for correspondence and other forms of communications, accounting, duplicating, filing, etc., it fulfills its service function. The *office mission* defines the responsibilities of the office in control and service areas of the business. Management is the coordinating factor which brings together the elements of an office in a harmonious unit to accomplish the office mission. Office management coordinates (1) *personnel*, (2) *methods*, and (3) *facilities* within the office. An *office organization* is created to define the job and job relationships of each office employee. This organization is hierarchical with management personnel at the top and office workers (professional and support personnel) at the bottom. Because of problems created in horizontal information flow through a vertical hierarchical organization, a set of *office procedures* defines the methodology of information flow. An office procedure does not provide methods for any specific work activity; but rather, provides methods for general information movement without regard to a specific work assignment. Once we have a mission, organization and procedures; we still are not in business. We must have a *people/facilities assignment*. Facilities must be assigned for employee use. A one-to-one correspondence is not necessary, as many people may use one machine (i.e. a self-service duplicator) or many facilities may be assigned one individual (i.e. multiple file cabinets). Associated with facilities assignment is the office layout. Layout is the allocation of space to departments and individuals. Finally, we must create *exo-office interfaces and procedures*. Since the office acts as the control and service hub of a business, methods must be established for communication outside the office. Some communications may stay with a business (office to factory communication) and some may move outside a business (business to business correspondence). Because of the sensitivity of this information flow, approvals within the hierarchical office organization are needed.

Our next question is: What is characteristic of this office *with regard* to specific work? We have a framework within which work can be completed, but we haven't established any specific work to be done or how it is to be controlled. *Workflow* is the movement of specific work through the framework created by an office mission, office organization, office procedures, people/facilities assignment, and exo-office interfaces and procedures. The framework is not sufficient to fully characterize workflow control. One of the functions of management in an office is to define work activities consistent with the office mission. This function is *work def-*

*inition*. Once a specific work activity is decided upon *work assignments* are given to the principals involved. The work assignment requires knowledge of the specific *work organization* (principals and support) for carrying out the work activity. The work assignment initiates work activities. The work activity is broken by the principals into a prescribed set of office processes. Each informational datum associated with the work is associated with a stage of completion in each office process. As the datum moves through the office process, it progresses from process stage to process stage. This progression is called *work staging*. Within each work stage, work activity must progress in a timely fashion. In addition, the stages must progress for each datum in a timely fashion. This relationship of time to work staging and activities in a stage is called *work scheduling*. Work does not always progress as planned. Corrective actions must be taken during many work activities to bring the activity in line with management expectations. This corrective function is called *work enforcement*. Timely control and management of work require a complete knowledge of the status of the work activity. To obtain this knowledge, *work measurement* is required. Measures are set up to gauge work progress and to compare with pre-established work criteria.

OFFICE MODEL/ELEMENT RELATIONSHIPS

The workflow control model ( $P_c''$ ) is a Petri Net [4,12] constructed in the knowledge domain ( $Q$ ) of knowledge states, state transitions, marking tokens, and process domain production rules. To make the elements of workflow control useful, we must be able to define each element in terms of the constituents of the  $P_c''$  model. In order to create these definitions, we must first develop a definition of the work information data which are manipulated in process domain

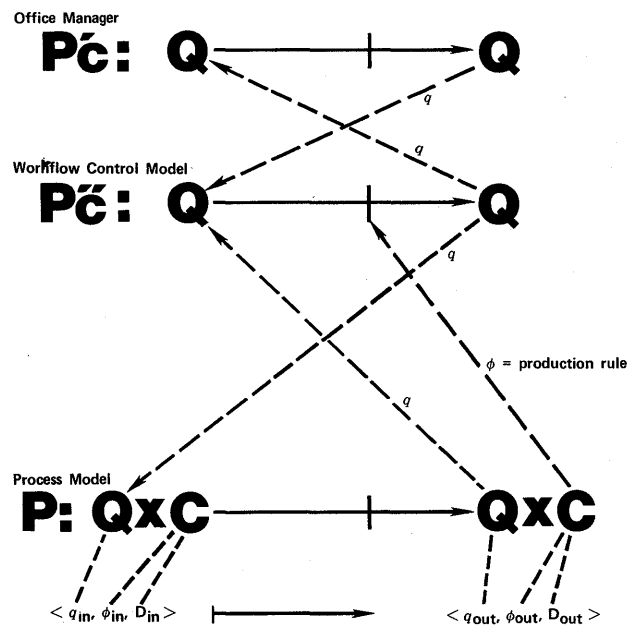


Figure 3—Model interconnections.

$(Q \times C)$  by the manual workers ( $P_i$ ), mechanized workers ( $P_j$ ), and mechanized subprocesses ( $P_{\sigma}$ ). The elements of the cartesian cross product  $(Q \times C)$  are triplets  $\langle q, \phi, D \rangle$  where:

- $q$  is the knowledge domain state of the datum. This state answers the "what" and "when" questions about the documentation.
- $\phi$  is the location-owner state (file, workstation, worker, etc.) of this datum. This state answers the "who" and "where" questions about the documentation.
- $D$  is the documentation of this datum.

The resultant knowledge state ( $q_{out}$ ) in the workflow control model ( $P_c$ ) provides the input state ( $q_{in}$ ) in the process models ( $P_i$ ), ( $P_j$ ), and ( $P_{\sigma}$ ). This input process state ( $q_{in}, \phi_{in}$ ) references the documentation triplets  $\langle q_{in}, \phi_{in}, D_{in} \rangle$  for processing. The resultant process state ( $q_{out}, \phi_{out}$ ) of a process model relates to the model processed documentation triplet  $\langle q_{out}, \phi_{out}, D_{out} \rangle$ . The resultant process state ( $q_{out}$ ) of a process model provides the input state ( $q_{in}$ ) in the workflow control model. The resultant process state ( $\phi_{out}$ ) of a process model provides a production rule ( $\phi_{in}$ ) in the work-

flow control model. This construction can be visualized as shown in Figure 3.

Our next step is to conceptualize the  $P_c$  workflow control model as a series of models constructed around each of the elements of workflow control. The models will be:

- $P_1$  ~ Office Mission
- $P_2$  ~ Office Organization
- $P_3$  ~ Office Procedures
- $P_4$  ~ People/Facilities Assignment
- $P_5$  ~ Exo-Office Interfaces & Procedures
- $P_6$  ~ Work Definition
- $P_7$  ~ Work Organization
- $P_8$  ~ Work Assignment
- $P_9$  ~ Work Staging
- $P_{10}$  ~ Work Scheduling
- $P_{11}$  ~ Work Enforcement
- $P_{12}$  ~ Work Measurement

To complete our construction of  $P_c$  we need another model  $P_0$  which relates the workflow control elements together. The  $P_c$  model would become:

$$P_c = \{P_i\}, i = 0, 1, \dots, 12$$

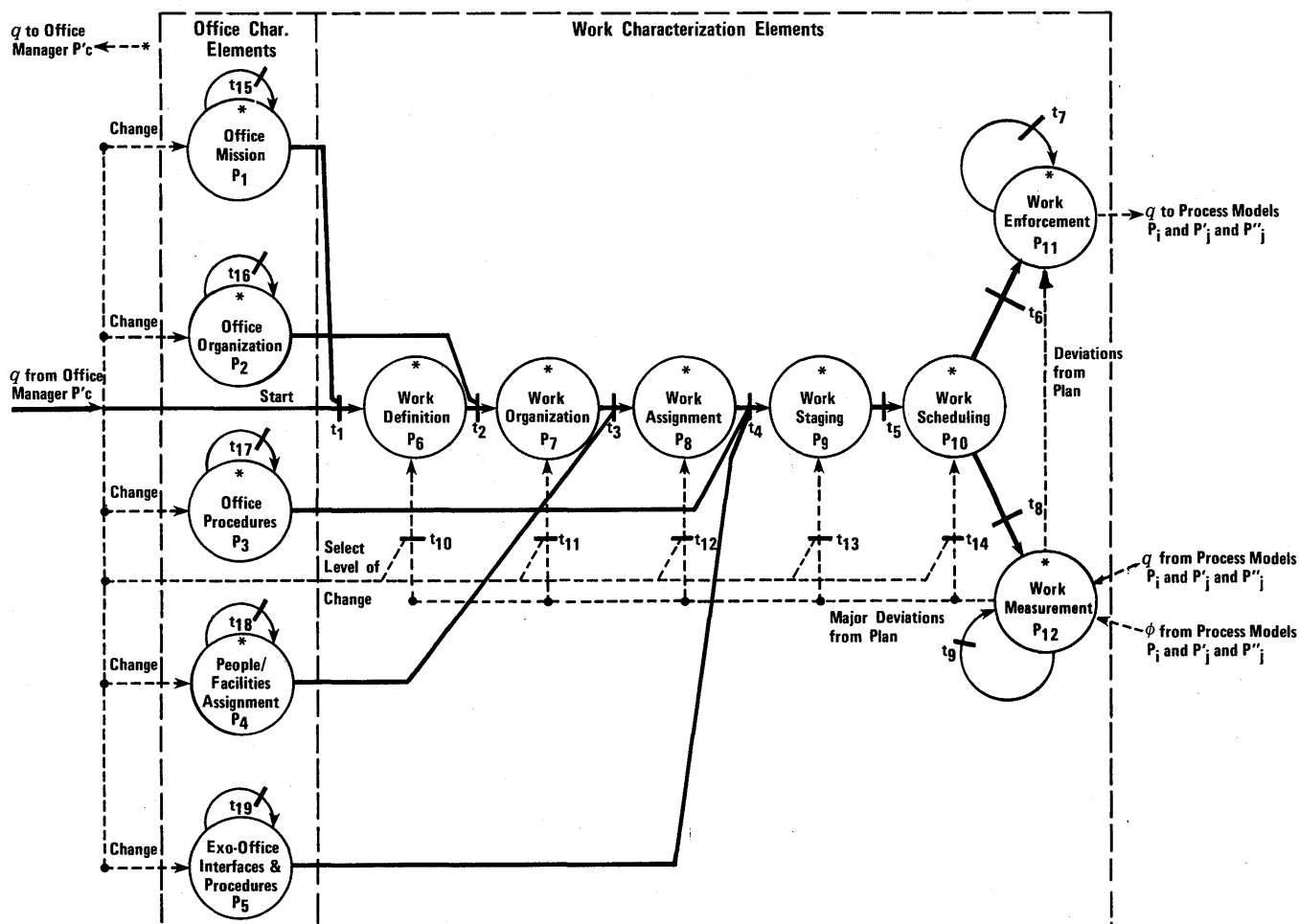


Figure 4—First layer ( $P_0$ ) of workflow control model ( $P_c$ ).

The first layer,  $P_0$ , of this new construction is shown in Figure 4. This model is a basic Petri Net with states  $P_1$  through  $P_{12}$ . These states act as triggers to initiate the corresponding lower level layer. For example, state  $P_i$  in the  $P_0$  net triggers the  $P_i$  net. Completion of the  $P_i$  net activates the transition following the  $P_i$  state in the  $P_0$  net. The solid lines indicate transition paths on the  $P_0$  net and the dashed lines indicate paths in the lower level  $P_i$ ,  $i=1,2,\dots,12$  nets.

Consider the right hand side of the  $P_0$  net. The exchanges with the Process Models are shown. Work Enforcement,  $P_{11}$ , is continually active and issues process initiation and change information ( $q$ ) to the Process Models. Changes are determined when deviations are noted by Work Measurement,  $P_{12}$ . Work Measurement compares the fixed work schedule with  $q$ ,  $\phi$  data received from the Process Models. Major deviations from plan are relayed to the Office Manager and changes can be made at the  $P_6$ ,  $P_7$ ,  $P_8$ ,  $P_9$ , or  $P_{10}$  layers as determined by input from the Office Manager.

Looking at the left hand side of the  $P_0$  net, we see the exchanges with the Office Manager,  $P_c'$ . The input from the Office Manager may start a work activity, via transition  $t_1$ , or change office characterizations, via  $P_1$ ,  $P_2$ ,  $P_3$ ,  $P_4$ , or  $P_5$  layers, or select changes in work characterizations, via the  $P_6$ ,  $P_7$ ,  $P_8$ ,  $P_9$ , and  $P_{10}$  layers. The output to the Office Manager consists of state information  $q$  from each of the  $P_c''$  layers.

The variation in office structures is incorporated in the office characterization processes  $P_1$  through  $P_5$ . These processes may be viewed as control structure processors which convert office characterization statements into office structure states for use by the work characterization processes  $P_6$  through  $P_{12}$ . Work characterization processes  $P_6$  through  $P_{10}$  provide the decomposition of a work activity selected by the Office Manager into specifically scheduled personal processes to be carried out by workers and process mechanizations. Work characterizations processes  $P_{11}$  and  $P_{12}$  provide direct control of workers and mechanized processes on a continuous basis.

## CONCLUSION

This paper has demonstrated that workflow control exists as an isolatable and mechanizable subprocess in every office

process. An office process model was created which isolates workflow control from the individual personal processes which constitute the office process. The workflow control mechanization provides for efficient work interchange and allows for interactive management control of work. Personal process mechanization improves support responsiveness. Based on the previous studies, this mechanization of workflow control and selected personal processes will improve office productivity significantly.

The models developed in this paper consider only a single control span within the office organizational hierarchy. Multiple spans of control in the hierarchy are obtained by interconnection of the independent workflow control models, and decomposing work definition in accordance with organizational charters. Matrix organizational structures are accommodated by partitioning the definition of office manager into functional manager and project manager constituents.

## BIBLIOGRAPHY

1. Statistical Abstract of the United States, 1977.
2. Lodahl, T. M., et al., *Providing Management Support in the Automated Office*, Corporate Systems, June 1979.
3. Steinfatt, T. M., *Human Communication*, Bobbs-Merrill Educational Publishing, 1977.
4. Zisman, M. D., *Representation, Specification, and Automation of Office Procedures*, Wharton School Working Paper 77-09-04.
5. Wetzler, R., "Operational Analysis—A Team Approach to Productivity Improvement," *Bests Review*, March 1978.
6. Mintzberg, H., "The Manager's Job: Folklore and Fact," *Harvard Business Review*, July-August 1975.
7. Blair, J., "Productivity Assessment of Office Information Systems Technology," *Trends and Applications*, 1978 Distributed Processing Conference.
8. Aiken, W. and Lewis, J., *Office Work Measurement*, Handbook of Business Administration, McGraw-Hill Book Company, 1970.
9. Zisman, M., "Office Automation: Revolution or Evolution," *Sloan Management Review*, Spring 1978.
10. Steely, J., "When Management is Automated," *Datamation*, April 1978.
11. Gibson, R., "Increasing Employee Productivity," *AMACON*, 1976.
12. Kaplan, M. and Schwartz, S., *Human Judgement and Decision Process in Applied Settings*, Academic Press, 1977.
13. Ness, D., *A Family of Systems which Process Semi-Structural Information*, Wharton School Working Paper 75-06-08.
14. Peterson, J., "Petri Nets," *Association of Computing Machinery-Computing Surveys*, September 1977.

# Streamlining office procedures—An analysis using the information control net model

by CAROLYN L. COOK

Xerox Palo Alto Research Center  
Palo Alto, California

## INTRODUCTION

The purpose of this paper is to acquaint the reader with a model for office procedures, the *Information Control Net model*, and a particular type of transformation that can be performed on an Information Control Net (ICN) model, *streamlining*. The ICN formalism is intended to aid office managers and office analysts in describing and evaluating procedures. Streamlining is a technique for reducing the ICN model of a procedure to a model of the necessary information flow and elementary information-processing of the procedure. Streamlining highlights the origin and destination of information in a procedure and allows the modeler to vary the route the information takes. Streamlining an ICN model of a procedure illuminates information-processing needs, activity by activity, in a way that may be useful for evaluating or changing the original procedure.

The model described here is based on a theory of Information Control Nets, which are directed graphs of procedures.<sup>1</sup> Information Control Nets (ICNs) partition the structure of a procedure and the information used in a procedure. This distinguishes ICNs from the traditional flowchart models of procedures in which communication was not explicitly represented. ICNs have nodes corresponding to *activities* that comprise a procedure and nodes corresponding to *repositories* (databases) used during a procedure. To illustrate an ICN model and the technique for streamlining, an ICN model of an office procedure in a savings bank is presented here as an example.

### *Using an ICN model*

The ICN formalism is a tool for describing office procedures. An ICN model is an instrument for evaluating and constructing alternative procedures. An ICN model enables the evaluation of the *control structure*, or organization of activities, and the *information structure*, or communication and use of information, in a procedure. An office analyst interested in the control structure of a procedure could look at an ICN model and see, for example, the activities that comprise a loop, and the branches of a procedure that follow a decision. An ICN model also shows the opportunities for

executing activities in parallel. An analyst interested in the information structure of a procedure could see the information requirements of specific activities, and patterns of access of office databases. The analyst could also see communications patterns, both within and among offices. After transforming an ICN model via streamlining, an analyst may view these same characteristics of the streamlined procedure, which emphasizes information requirements, and not the logistics of how the requirements have been met.

Probabilities may be used to label control flow arcs when a procedure includes choice nodes (decisions or or-splits). Execution time may be added to the model as an attribute of activities. Handling probabilities and execution times is not dealt with in this paper, although they are within the scope of the model.

An ICN model also provides a framework for analyzing office procedures. For example, an ICN model would allow an office manager to consider the appropriateness of different media for office communications based on inter- and intra-departmental communications patterns. Or the manager could compare a paper system to an electronic system. Alternative procedures may be suggested by streamlining an ICN model; one large form may be divided into two smaller forms, each with a different route. While the ICN model presented here gives a static description of a procedure, an ICN model can also be the basis for dynamic simulations to allow computer-aided, interactive analysis of offices.<sup>2,3,4</sup>

Like any model, an ICN model cannot reflect everything that happens in an office. An ICN model does not reflect the intent of a procedure. An ICN model does not indicate specific optimizations of a procedure. Despite these limitations, the model captures the organization of a procedure and the patterns of information usage in the office well.

The basic representation of an ICN model is a diagram of an office procedure. The diagrams can be understood and verified by officeworkers because the notation lends itself to intuitive analysis. The diagrammatic model is based on a formal mathematical ICN model. Because an ICN model focuses on relationships among activities and information in a procedure, these relationships can be viewed independent of how the activities are executed and what the content of the information communicated is.

As a procedural notation, an ICN model allows an office



analyst to collect and compare descriptions of the same procedure. The analyst may construct a diagram which can be compared either to the office manager's overview of the procedure or to each officeworker's perception of the procedure. With this model, inconsistencies in the abstract formulation of the procedure as well as between different interpretations of the procedure are more readily discernible. Alternative procedures can be shown to the officeworkers and office manager to discuss and evaluate.

*Streamlining*

There are many possible permutations of a streamlined ICN model. Different permutations result from using different mapping schemes to relabel repositories (databases). After a mapping scheme has been selected, the streamlining transformation produces a streamlined ICN model of the original procedure.<sup>5</sup> For example, one mapping scheme for repositories is to aggregate them into fewer repositories of differing type; archives, references, log books. This mapping scheme encourages the modeler to look at generic activities. Another mapping scheme is separation of repositories into many more specialized ones. This mapping scheme corresponds to the unbinding of forms and files and encourages the modeler to treat different types of information on one form differently. The streamlined ICN model is not intended to be installed in an office, rather, it is meant to suggest ways to restructure the original procedure.

CONSTRUCTING AN ICN MODEL

An ICN model of an office procedure can be expressed as a graph or diagram. A procedure is represented as a set of activities embedded in a *control structure*, which shows the temporal ordering of activities. Activities are left uninterpreted in the model; the modeler represents how activities fit into the control structure of the procedure, but does not represent how they are executed. Graphically overlaid on the control structure diagram is the corresponding *information structure* diagram. The information structure shows the use of forms, files, and databases where information is stored during the procedure. Control structure or *control flow* is diagrammed in bold-faced lines; communication or *information flow* is diagrammed in light-faced lines. There are data labels associated with each of the information flow lines. The basic notation used in ICN models is bold-faced arcs and circles for the control structure, and light-faced arcs and polygons for the information structure.

The information and control structures of a procedure as represented in an ICN model may be implemented in a variety of ways. Information-handling may be implemented electronically as well as with paper forms and verbal communication. The execution of activities may be implemented by a group of officeworkers or entirely by one officeworker. The implementation of the procedure (how it is done, when, and by whom) is not expressed in an ICN model. Rather,

different implementations may be evaluated within the framework of an ICN model.

*Elements of the ICN model*








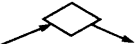



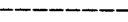
**Activities**

A procedure is modeled as a set of activities executed in a specified order. There is nothing inherent in the ICN formalism that dictates the level of detail of a task represented by an activity. Activities represent discrete operations (for example, office tasks) and have as attributes a descriptive label, a place in the control structure, a set of *repositories* (databases) which it may access, and information input(s) and output(s). An activity may be called *elementary* if it accesses at most one permanent repository.

**Repositories**

A repository in an ICN model denotes a storage site of information communicated between activities. Repositories are used to denote documents, files, forms, record books, scratch paper, and people's heads. The label for a repository corresponds to the physical presence of information at a storage site. The representation of storage sites is somewhat ab-

TABLE I—Legend

	<b>Control Flow Arc</b>
	<b>Activity</b>
	<b>Decision Node ("or" - split)</b>
	<b>"Or" - join</b>
	<b>"And" - split</b>
	<b>"And" - join</b>
	<b>Information Flow Arc</b>
	<b>External Permanent Repository</b>
	<b>Internal Permanent Repository</b>
	<b>Temporary Repository</b>
	<b>Common Temporary Repository</b>
	<b>Departmental Boundaries</b>
<b>p s</b>	<b>Data Labels</b>

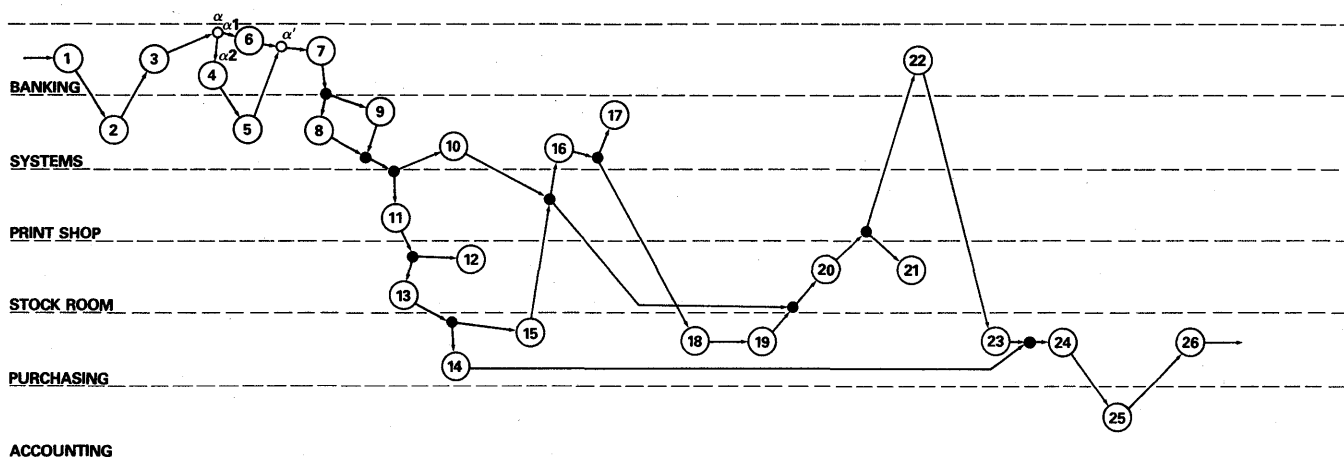


Figure 1.

stract, files containing forms are modeled as a single repository for the information on the forms, and not a repository within a repository. There are four kinds of repositories in the ICN formalism. *External permanent repositories* represent a class not to be restructured or coalesced, for example, a person, or the *Wall Street Journal*. *Internal permanent repositories* represent permanent office records and databases, for example, a record of customer accounts. They may be accessed outside the scope of one procedure. *Temporary repositories* represent working files containing infor-

mation of topical interest or immediate usefulness, e.g., a piece of scratch paper containing a number, or a telephone message to return a call. *Common temporary repositories* represent the merger of all temporary repositories, such that information stored into any one of them may subsequently be retrieved from any other one. Common temporary repositories have some of the characteristics of multi-access electronic databases, a useful concept for office information system design. In the ICN notation, external permanent repositories are represented by diamonds; internal permanent

TABLE II—Directory of Activities

<u>Activity Number</u>	<u>Activity Description</u>	<u>Activity Number</u>	<u>Activity Description</u>
1	Write Request	16	Fill Out Supply Request Form
2	Produce Proof	17	File 1 Part Supply Request & 1 Sample Form
3	Decide Whether Proof is Adequate	18	Authorize Supply Request Form
4	Amend Proof	19	Log Supply Request Form
5	Produce Final Design of Proof	20	Fill Request
6	Designate Proof Design as Final	21	File 1 Copy Supply Request Form
7	Approve Proof	22	Sign Supply Request Form
8	Copy Proof	23	Decrement Inventory by Quantity of Forms Delivered
9	Fill Out Form Preparation Sheet	24	Calculate Chargeback Costs
10	File 1 Copy Form Preparation Sheet & 1 Copy Proof	25	Post Charges Against Originator's Budget Center
11	Print Forms	26	File Supply Request Form
12	Place Forms in Open Stock		
13	Distribute 2 Sample Forms & 1 Copy Form Preparation Sheet		
14	File 1 Sample Form & 1 Part Form Preparation Sheet		
15	Make Up Inventory Card		
		<u>Decision</u>	<u>Decision Description</u>
		$\alpha$	Decision to Accept Proof
		$\alpha 1$	Decision Branch if Accepted
		$\alpha 2$	Decision Branch if not Accepted
		$\alpha'$	Or-juncture, reuniting branches $\alpha 1$ and $\alpha 2$

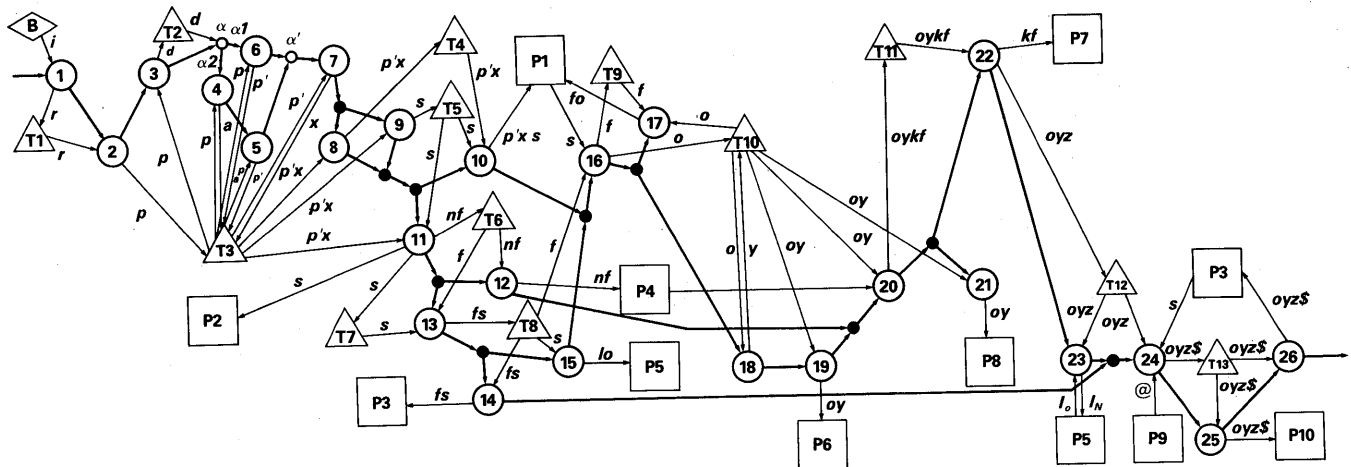


Figure 2.

repositories by squares; temporary repositories by triangles; and common temporary repositories by shaded triangles.

**Control flow arcs**

Control flow arcs represent the precedence constraints among activities. The beginning of a procedure is represented by a bold-faced arc leading into an initial activity (some procedures may have more than one). The termination of a procedure is represented by a bold-faced arc extending from a terminal activity. A control flow arc between two activities denotes that the first must be completed before the second can begin.

Two procedural characteristics that can be represented easily in an ICN model are parallelism and choice. In some procedures there are cases of no precedence constraints among a set of activities; the activities may be executed in any order or simultaneously. The modeler might like to reflect this freedom of control flow in an ICN model.

**And-splits and and-joins to reflect parallelism**

Parallelism in a procedure may be represented by inserting a juncture (a black dot) connecting multiple arcs. When an activity preceding an *and-split* (a black dot) has been completed, any or all of the activities immediately following the

TABLE III—Directory of Repositories

<u>Repository Number</u>	<u>Repository Description</u>	<u>Repository Number</u>	<u>Repository Description</u>
<b>External Permanent Repositories</b>		<b>Temporary Repositories</b>	
B	Banker	T1	Request for New Form
<b>Internal Permanent Repositories</b>		T2	Decision About Proof
P1	Systems File	T3	Proof File
P2	Print Shop File	T4	Internal Systems File
P3	Purchasing File	T5	Form Preparation Sheet
P4	Open Stock	T6	Printed Forms File
P5	Inventory	T7	Stock Room In-Box
P6	Supply Log	T8	Stock Room Out-Box
P7	Banking Supplies	T9	Internal Systems File
P8	Stock Room File	T10	Supply Request Form
P9	Cost Data	T11	Banking In-Box
P10	Accounting Records	T12	Purchasing In-Box
		T13	Purchasing Out-Box

TABLE IV—Directory of Data Labels

<u>Data Labels</u>	<u>Information</u>
i	idea to request new form
r	request for new form
p	proof
a	amendments to proof
p'	final proof
d	decision about proof
x	approval of proof
s	specifications for form
f	form
n	number of forms printed
k	number of forms ordered
o	order for forms
y	authorization for supply request form
z	signature from banking
@	cost per form
\$	cost of forms
l <sub>o</sub>	old inventory of form
l <sub>n</sub>	new inventory of form

and-split may be initiated. An *and-join* (also a black dot) may also be used to join control flow arcs to denote that all of the activities preceding the and-join must be completed before the next activity can be started. The passing of control beyond the and-join must wait for the longest or slowest branch.

#### Decisions, or-splits, and or-joins

To represent alternative branches or loops with the ICN formalism, the alternative branches of activities are modeled following a hollow dot in the control structure. This hollow dot represents an *or-split*, which is a very simple activity. An or-split represents a branch point where the determinants of which branch is followed are (or may be modeled as being) probabilistic and nondeterministic. Occasionally in a procedure, a decision is made to determine which of several alternative branches will be followed. A *decision* is a special case of or-split, where the choice among alternative branches depends on information from the activity preceding the or-split. Decisions are labeled, as are the several arcs that diverge from them (e.g.  $\alpha$  labels a decision,  $\alpha_1$  and  $\alpha_2$  identify two branches following  $\alpha$ ). Control flow arcs converge at an *or-join*. An or-join is used as a shorthand to denote identical segments of a procedure which follow different branches. The activity following an or-join may have been preceded by any of the activities immediately preceding the or-join.

#### Information flow arcs

Communication and data storage and retrieval in a procedure are represented by *information flow arcs* (light-faced arcs) between the activities which generate or use the information, and the repositories where information is stored between activities. The information flow arcs in an ICN model reflect possible routes of information flow rather than necessary ones. All information used or produced by the activities is represented in the ICN formalism as *data labels*. These labels designate the information transferred between activities and repositories. A directory of data labels and the information they represent must be generated by the modeler for each ICN model.

Each piece of information can be considered to have a source activity (point of origin or retrieval) and a sink activity (point of use or storage) in the procedure. Some information may have multiple sources and multiple sinks. Between its source(s) and sink(s), information may be stored into and retrieved from temporary repositories by any number of activities.

#### Departmental boundaries

An ICN diagram can be drawn so the placement of activities corresponds to some particular characteristic of the activities. For example, activities may be grouped in horizontal bands corresponding to the departments of a company, or to individual officeworkers in an office. In the example developed in this paper, *departmental boundaries* were represented by dashed lines. Each activity fell into one of six departments and was drawn within the horizontal band corresponding to that department. An ICN diagram could also be drawn so the placement of activities corresponded to the time required to execute them, with vertical bands denoting the elapsed time from the start of a procedure to the end of it. The modeling of elapsed time is not discussed in this paper.

#### Data requirements for the ICN model

The first requirement in discerning what information is relevant is an understanding of the procedure to be modeled. When the scope of a procedure has been determined, model construction involves a judgment about the level of specificity for the activities. The judgment is not so much "At what level of detail does this procedure exist?" but "At what level of detail do we want to look at this procedure?" The decision may be based on the level at which a manager is willing to restructure the procedure, or the level of detail that provides a view of as much communication between two offices as possible.

The information best represented by an ICN model is an operational description of what each officeworker or department does and what information is used or produced in the course of the procedure. Knowing who in the office is responsible for which activities and where the activities are

executed permits more sophisticated interpretation of an ICN model. Knowing the tasks that comprise each activity gives the modeler additional insight to choose the most useful level of detail for his/her purposes. A modeler hopes for information of this type: "When a person calls to request a new form, I write down his name, department, and the kind of form he has in mind. I ask if any form we already have resembles the one he wants. The designer likes to know this. Then I make an appointment for a forms designer to meet with him to sketch out a prototype. So I keep the appointment books of all of the forms designers."

It is most important to note the exact information needed for each activity, the databases used, the order in which things are done and, when discernible, the reason for that order. It is useful to note the media used for all communications, the method of handling errors and special cases in an activity, and the choices or decisions made by the office-worker. Measurements of the time typically devoted to each activity and the probabilities that a decision will result in one alternative or another can be evaluated in the framework of an ICN model. An interesting source of information is a list of the problems encountered by officeworkers in carrying out an activity. These may suggest areas where analysis will be particularly helpful.

The methodology for constructing an ICN model has been to observe an office, define a procedure to be modeled, and interview the officeworkers responsible for that procedure. After a diagrammatic model has been produced from the collective interviews, it can be checked for self-consistency, and shown to the officeworkers and office manager for verification. The process is iterative rather than sequential; an incomplete or inconsistent diagram would require further interviewing and observation, and that in turn would result in another diagram.

A characteristic of ICN models that requires some deliberation is the modeler's definition of the scope of the procedure. Identifying the procedure to be modeled is a choice the modeler must make early in the construction of the model, and has significant implications for the level of detail captured by activities, repositories, and data labels. The modeler's discretion is exercised early and the wisdom of his/her choice is realized later, which suggests that several iterations of model-building should be expected.

#### EXAMPLE: A PROCEDURE TO GENERATE NEW FORMS

##### *Overview of the new forms generation procedure*

The procedure selected for this modeling exercise is one followed by a savings bank to design new forms on demand and make them available for internal use. The data describing this procedure had been gathered for another study, and suited an ICN model well. Some assumptions were made about the new forms generation procedure for purposes of expediting research when the data were ambiguous or absent. A detailed description is available of the original pro-

cedure and the assumptions made to construct this ICN model.<sup>6</sup>

The new forms generation procedure is carried out mainly by the Systems Department of the bank, although six departments are involved in the procedure [Figures 1,2, Tables I,II,III,IV]. The procedure is initiated when a functional department of the bank finds the need for a form, and no existing form seems adequate. Each instance of the procedure starts with a request from some department desiring a new form to a member of the Systems Department.

After a prototype form has been designed by the Systems Department and approved by the requestor, the Systems Department arranges to have the new forms printed at the Print Shop. After the new forms are printed, they are sent to a Stock Room where some remain as open stock and some are sent to the original requestor (in response to the initial request for a new form). The mechanism for sending the newly printed forms to the requestor appears to be a special case of a routine supply request. The Systems Department fills out a request form on behalf of the requestor, then processes that request form as they would one for any stocked item. The Accounting and Purchasing Departments are notified of the request and response. The Purchasing Department calculates the expense to the requestor of new forms printed and both departments make the appropriate changes to inventory and accounting records.

##### *Construction of an ICN model of the procedure*

The construction of an ICN model is an iterative process. The activities represented in an ICN model describe the procedure from start to finish. While there need not be a discrete activity for each task, each task must be associated with some activity. All data used by an activity are modeled as being read from a repository, and all data produced by an activity are modeled as being stored in a repository. Information is not sent directly from one activity to another, rather it is stored in a repository where it can be accessed by one or more activities. Data stored in a repository can be read from that repository more than once and by more than one activity. Unless otherwise stated, the process of reading information from a repository is non-destructive; information is retrieved for activities as if only copies can be retrieved.

The first task in constructing an ICN model was to identify the activities that made up the procedure and to establish the precedence relationships among them. Each time consecutive activities took place in different departments, information was sent from one department to the other. "Send" and "receive" activities are not modeled explicitly. An information flow arc between an activity and a repository indicates that information has been sent or received. When activities were ordered for no apparent reason, the order was preserved. However, when activities were executed in parallel or in either order, they were modeled as concurrent using the and-split and and-join notation.

The activities were diagrammed in a two-dimensional space where, along the horizontal axis, activities to the left

generally precede activities to the right, and along the vertical axis, each activity fell into one of six bands corresponding to the six functional departments. [Figure 1] Repositories and information flow lines appear in the style of an overlay, and their location on the ICN diagram was chosen for proximity to the activities using the data. Therefore, the location of repositories and information flow lines does not correspond to the functional department where the information is stored. The physical existence of information is treated abstractly in an ICN model. Information is modeled with no indication of medium. Rather, information is modeled to reflect content, via data labels; local or global relevance, according to whether it is stored in a temporary or permanent repository; and usage, according to its creation, storage, and usage by the activities comprising the procedure. Other attributes of the information structure must be interpreted from the model.

## STREAMLINING

An ICN model allows the office analyst to explore the consequences of changing the information and control structure of an office procedure. An ICN model may be subject to several transformations, of which streamlining is only one. A streamlined ICN model is the simplest representation of the points of origin, or *sources* and the uses or destinations, or *sinks*. The order of activities in the streamlined model is not changed. However, activities which neither produce nor require information (the middlemen activities, as it were) are not represented in the streamlined ICN model. An ICN model may also be subject to restructuring transformations which suggest ways of altering the procedural policy or redefining the domain of an office or department. Finally, an ICN model may be subject to automation transformations, wholly or partially, which suggest ways of executing a procedure in the environment of a computer-based office information system (OIS). An OIS may offer such capabilities as access to electronic databases, message systems, text editors (word processors), inventory control systems, inquiry systems, computer-aided forms fill-out, programs for data analysis, and decision support systems. A discussion of the restructuring and automation transformations is beyond the scope of this paper.

### *Purpose*

The purpose of streamlining is to provide an office analyst with different views of a procedure; views of how information is processed in the course of the procedure. By looking at the information-processing needs, an analyst can evaluate alternative ways to meet those needs by changing the information-handling characteristics of the original procedure. Streamlining reduces an ICN model to the basic communication and information requirements of an office procedure. Streamlining highlights the information-processing needs of a procedure by identifying the activities where information is originally created or retrieved and fi-

nally used or stored. Streamlining also allows variations in the information-handling characteristics by eliminating the activities where information is simply distributed rather than processed, and by coalescing activities whenever possible. The implementation of local communication is hidden in a streamlined ICN model, to give a more abstract and simpler view of where information is produced and where it is used. Communication is represented by effectively short-circuiting the information paths from creation to use.

### *Method*

The streamlining transformation produces a streamlined ICN model of the original procedure according to a repository mapping scheme. Streamlining is achieved by selecting a mapping scheme for repositories, then coalescing activities. Repository mapping frees the modeler from representing the organization of forms and files as they exist in the office.

Different permutations of streamlined ICN models result from using different mapping schemes to relabel repositories. The repository mapping scheme discussed here aggregates repositories according to type; archives, inventory records, cost information, and the common temporary repository. (External permanent repositories are never aggregated.) By collapsing all temporary repositories into one, an office analyst can focus on the sources and sinks of information independent of the path between them. This mapping scheme encourages the modeler to look at generic activities. An alternative mapping scheme is to separate repositories into many more specialized ones. This mapping scheme corresponds to the unbinding of forms and files and encourages the modeler to treat different types of information on one form differently.

### **Identifying sources and sinks of information**

Activities are sources and sinks with respect to particular pieces of information (*datum*). A datum (designated by a data label) may have two types of *source*. one type of source is an activity where the datum is created. These may be identified in an ICN model as every activity for which the datum appears on an information flow arc as the output of that activity, and was not present on an information flow arc which was the input of that activity. Another type of source is an activity where the datum was retrieved from a permanent repository and had not been stored in that permanent repository earlier in the procedure.

A datum may have more than one source. One circumstance of multiple sources occurs if there is more than one alternative path in the control structure from source to sink. The activity which is a source for a datum is designated not only by an activity number or label, but also by a specific path through the control structure from source to sink. Alternative paths through the control structure are identified by the labels following a decision activity ( $\alpha 1$ ,  $\alpha 2$ ).

A datum may have four types of *sink* in an office proce-

ture. It may have a sink in every activity where it was used to create a new datum. It may have a sink in every activity where it was used to make a decision. It may have a sink in every activity where it was stored in a permanent repository. It may have a sink in every activity where it was used as a key to retrieve other information. The fourth type of sink (a datum used as a key) requires a special notation, and is commonly denoted by an information flow arc with two small parallel lines intersecting it.

Each datum may be thought to have pairs of sources and sinks. In the case of multiple sources, care must be taken to match each sink with its proper source. Source and sink pairs may be identified in an ICN diagram by a continuous sequence of information flow arcs from the source activity to the sink activity, through repositories and intermediate activities.

#### Eliminating and coalescing activities

An activity is unnecessary for information-processing if it neither stores information permanently, nor creates new information, nor uses information as a key to access other information, nor determines a decision. Unnecessary activities may be eliminated in streamlining. Activities which perform redundant storing of information in one repository may be eliminated during streamlining. Once information is stored, it need not be stored again. Likewise, activities storing information in a temporary repository may be eliminated if the information is not subsequently retrieved from the temporary repository. If information has been stored in a temporary repository, and is subsequently read from a permanent repository, the retrieval activity in the streamlined model is transformed into a similar activity that retrieves the information from the temporary repository. The basis of this rule is that it is inefficient to retrieve information from a permanent repository when it was previously accessible in a temporary repository.

Activities may be coalesced after the unnecessary activities have been eliminated. Activities may be coalesced as long as they remain elementary activities (accessing no more than one permanent repository), and as long as no precedence constraints are violated. Activities may not be coalesced if they fall on different branches following an or-split. The branches may be streamlined independent of one another, and the entire procedure following an or-split may be replicated for each branch (rather than re-joined with an or-join) to view the branches following a decision node more distinctly. Old decision activities are mapped into identical new decision activities. A decision activity may not be coalesced with any other activity.

The fundamental information requirements preserved by streamlining relate an activity which is the source of a piece of information (by either creating the information or retrieving it from a permanent repository) to one of the sinks of that piece of information (because it has been either used or stored by the second activity). Because an ICN model may contain or-joins, there is sometimes more than one path to an activity. To uniquely identify an activity following an

or-join, the path to that activity is denoted by the label of the control branch following a decision activity ( $\alpha_1, \alpha_2$ ).

There are two determinants of the control structure (or precedence constraints) among activities. One is information requirements; the information produced in one activity must be available to the activities that require it (this condition will be specified more formally). The other determinant of the control structure is embedded in the procedure as a procedural technique. It may reflect corporate policy, local managerial preference, or office lore. This information is captured in the original ICN model of a procedure, and is generally preserved in the manipulations of an ICN model.

#### The streamlined ICN model

The streamlined model [Figure 3] was produced by mapping repositories and applying the rules for coalescing activities described above. The repository mapping scheme used to streamline the model of the new forms generation procedure [Table V] resulted in the elimination and coalescing of activities, or corresponding activity mapping scheme [Table VI]. Note that information stored in one common temporary repository may subsequently be retrieved from any other common temporary repository.

TABLE V—Repository Mapping Scheme

#### Permanent Repositories

P1 -->	A	Archive
P2 -->	A	Archive
P3 -->	A	Archive
P4 -->	O	Open Stock
P5 -->	I	Inventory
P6 -->	A	Archive
P7 -->	S	Supply
P8 -->	A	Archive
P9 -->	C	Cost data
P10 -->	R	Records - accounting

#### Temporary Repositories

T1 -->	T	T9 -->	T
T2 -->	T	T10 -->	T
T3 -->	T	T11 -->	T
T4 -->	T	T12 -->	T
T5 -->	T	T13 -->	T
T6 -->	T	T14 -->	T
T7 -->	T	T15 -->	T
T8 -->	T		

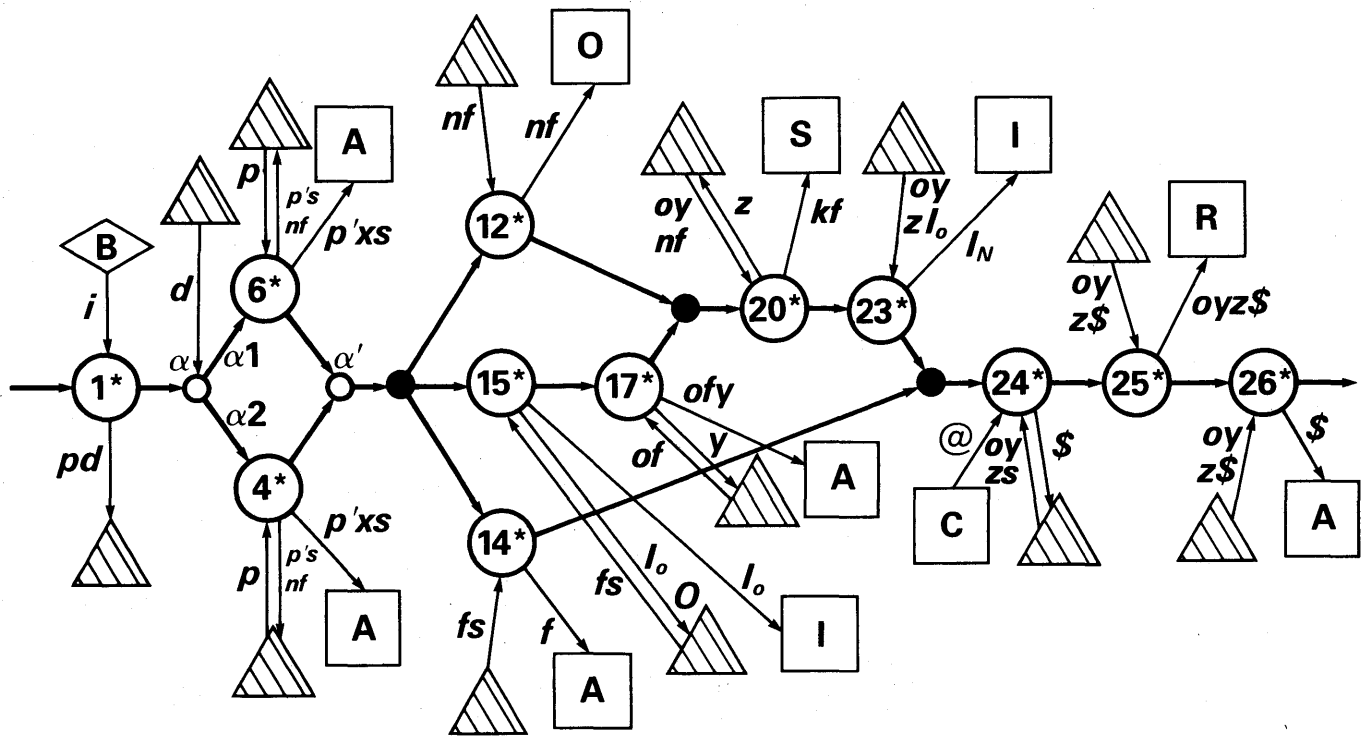


Figure 3.

The original 26 activities were streamlined to 12 elementary activities. Three activities were eliminated because they were neither sources nor sinks for any information (activities 8, 13, and 21). Of the remaining 23 activities, the greatest coalescing occurred in activities 4\* and 6\*, where the final proof design was submitted to the Print Shop. There was

moderate coalescing of the activities involving the translation of the idea for a new form to the proof (activity 1\*), and the approval and recording of the supply request form (activity 17\*).

After information *i* is used to create *p* and *d* (activity 1\*), the new forms generation procedure may follow one of two

TABLE VI—Activity Mapping Scheme

New Activity Number	Old Activity Number	Activity Description	New Activity Number	Old Activity Number	Activity Description
1*	<-- 1	Write Request	12*	<-- 12	Place Forms in Open Stock
1*	<-- 2	Produce Proof	null	<-- 13	Distribute 2 Sample Forms & 1 Copy Form Preparation Sheet
1*	<-- 3	Decide Whether Proof is Adequate	14*	<-- 14	File 1 Sample Form & 1 Part Form Preparation Sheet
$\alpha$	<-- $\alpha$	Decision Node (will branch to $\alpha 1$ or $\alpha 2$ )	15*	<-- 15	Make Up Inventory Card
4*	<-- 4	Amend Proof	15*	<-- 16	Fill Out Supply Request Form
4*	<-- 5	Produce Final Design of Proof	17*	<-- 17	File 1 Part Supply Request & 1 Sample Form
6*	<-- 6	Designate Proof Design as Final	17*	<-- 18	Authorize Supply Request Form
4*	<-- 7	( $\alpha 1$ branch) Approve Proof	17*	<-- 19	Log Supply Request Form
6*	<-- 7	( $\alpha 2$ branch) Approve Proof	20*	<-- 20	Fill Request
null	<-- 8	( $\alpha 1$ branch) Copy Proof	null	<-- 21	File 1 Copy Supply Request Form
null	<-- 8	( $\alpha 2$ branch) Copy Proof	20*	<-- 22	Sign Supply Request Form
4*	<-- 9	( $\alpha 1$ branch) Fill Out Form Preparation Sheet	23*	<-- 23	Decrement Inventory by Quantity of Forms Delivered
6*	<-- 9	( $\alpha 2$ branch) Fill Out Form Preparation Sheet	24*	<-- 24	Calculate Chargeback Costs
4*	<-- 10	( $\alpha 1$ branch) File 1 Copy Each Form Preparation Sheet & Proof	25*	<-- 25	Post Charges Against Originator's Budget Center
6*	<-- 10	( $\alpha 2$ branch) File 1 Copy Each Form Preparation Sheet & Proof	26*	<-- 26	File Supply Request Form
4*	<-- 11	( $\alpha 1$ branch) Print Forms			
6*	<-- 11	( $\alpha 2$ branch) Print Forms			



paths ( $\alpha 1$  or  $\alpha 2$ ). The two paths appear to be identical in the streamlined ICN model, and the choice of paths appears to be of no consequence to the rest of the procedure. The parallel structure of activities 12\*, 14\*, and 15\* reflect the order-independence of two filing activities and one information processing activity (activity 15\*) where the supply request form and inventory card ( $o$  and  $I_o$ ) are produced. After the supply request form is approved by both the Purchasing Department (activity 17\*) and the requestor's department (activity 20\*), inventory records are decremented. Only after the forms have been printed and delivered, and the records of five departments document production and delivery, is the cost calculated (activity 24\*), charged (activity 25\*), and recorded finally by the Purchasing Department (activity 26\*).

## ANALYSIS AND CONCLUSIONS

### *Analysis of the original ICN model*

Part of the value of an ICN model is the global perspective it can give. Of the six department managers who supervise the new forms generation procedure, only a few of them may understand the entire procedure. The department managers may not be aware of how many times the proof, the form being generated, and the supply request form are stored or retrieved (14, 15, and 16 times respectively). It is important that these three documents be designed to be understood quickly and processed with a minimum of errors. Communication between the requestor's department and the Systems Department is relatively intense in the first ten activities. This may suggest to the office analyst an opportunity to improve the procedure.

A glance at the ICN diagram of the new forms generation procedure [Figure 2] shows that most information is stored in and retrieved from temporary repositories in the earlier part of the procedure. This trend changes toward the middle of the procedure; information is more often retrieved from temporary repositories and stored in permanent repositories. Activities in the latter part of the procedure form a record for the early part.

Of the 26 activities in the new forms generation procedure, in only one instance are three consecutive activities in one department (activities 3, 6, and 7, Figure 1). This indicates that forms processing for this procedure is widely distributed. Because control is transferred so freely from one department to another, the procedure is potentially difficult to manage. This may explain why some activities are executed in sequence when the information is available for them to be executed concurrently. For example, after activity 11, activities 12, 13, 14, 15, and 16 could be executed concurrently. Activities 19, 20, and 21 are also candidates for concurrent execution, as are activities 23 and 24, and activities 25 and 26.

There appears to be redundant storage of information in internal permanent repositories. [Figure 2] For example,  $s$  (the specifications for the form) is stored in P1, P2, and P3, and  $o$  (the supply request form) is stored in P1, P3, P6, P8,

and P10. This redundancy might be a deliberate technique to minimize errors or expedite error-handling. An ICN diagram encourages managers to consider such characteristics of a procedure.

A two-dimensional matrix can be drawn from an ICN diagram with the originators of information (by department) along one axis and the recipients or users of information along the other axis. Using a matrix of this type, an office analyst can see the patterns of communication between and within departments. In spite of the interdepartmental nature of the procedure, it appears that more communication takes place within departments than between them. Most remarkable for interdepartmental communication in this procedure are the Banking Department, the Systems Department, and the Purchasing Department. Most interdepartmental communication occurs between the Banking and Systems Departments and the Banking and Purchasing Departments. With this information, the bank might decide to install an experimental electronic message system in either the Systems or Purchasing Departments.

Viewing the ICN model [Figure 2], an office analyst may come up with alternative communications media to be used in the early part of the procedure (activities 1 through 9), such as an aid for designing forms that is portable. A portable design tool might save time, reduce the probability of re-designing the proof (traversing path  $\alpha 2$ ), and encourage better design of forms.

### *Analysis of the streamlined ICN model*

In the streamlined ICN model, concurrency in the control structure of the procedure seems more obvious. [Figure 3] Looking at the basic information structure, the only permanent repository that has to be accessed as an information source is "C" (formerly P9, the cost data used by the Accounting Department). All other permanent repositories are information sinks; they are used for storage, primarily filing. An analyst may conclude that many of these filing activities (especially activities 12\*, 14\*, 25\*, and 26\*) may be executed more leisurely than activity 24\*. Activities 12\* and 14\* appear to be filing activities of a housekeeping nature in the Stock Room and the Purchasing Department. Of the three parallel branches following  $\alpha'$  (an or-join), the control path including activities 15\* and 17\* appears to be more critical to the completion of the procedure than either of the concurrent paths. In activities 15\* and 17,\* two documents are created, the supply request form  $o$  and the inventory card  $I_o$ . Both of these documents are necessary to the procedure.

The constraints that activity 12\* must be completed before activity 20\* is started, and activity 14\* completed before activity 24\* is started exist apparently for ease of management, since there is no informational requirement for their order. If it were necessary to rush through the procedure, these filing activities could be delayed. However, if the office manager's objective were to reduce the execution time of the procedure, the communication media for temporary repositories might be most important to optimize.

### Conclusions

The streamlining transformation is potentially valuable to an office manager responsible for managing lengthy or complicated procedures. A streamlined ICN model provides a compact survey of a procedure, and the model can be interpreted partially by inspection. An ICN model also is a framework for more sophisticated interpretation according to the manager's needs and interests. An ICN model may enhance a procedures manual by showing the relationships among activities in a notation which is more consistent than a verbal description and is easier to check for completeness. An ICN model is also a useful representation of a procedure to purchasing agents and division managers who may be interested in evaluating the need for office equipment and proposing changes in the policy underlying the office procedures.

### ACKNOWLEDGMENTS

The research reported here is part of an office modeling and simulation project being conducted in the Office Research Group, Xerox Palo Alto Research Center. The ICN model described herein was developed by members of the Office Research Group and the Analysis Research Group at P.A.R.C.

The author wishes to acknowledge her indebtedness to members of the Office Research Group and J. F. Rulifson for their criticism and guidance in this endeavor. In particular, Clarence Ellis has patiently advised the author and consulted with her in matters of great detail as well as great abstraction. It is my pleasure to thank these people for providing a stimulating working environment.

### REFERENCES

1. Ellis, Clarence A., "Information Control Nets," *Proceedings of the ACM Conference on Simulation, Measurement, and Modeling*, Boulder, Colorado, August 1979.
2. Nutt, G. J., "Backtalk; An office environment simulator," *1979 International Communications Conference*, Boston, Massachusetts, June 1979.
3. Nutt, G. J., "Modeling Office Information Systems," talk given at computer science seminar, University of California-Davis, Livermore, California, December 6, 1979.
4. Nutt, G. J., "An Experimental Distributed Modeling System," Xerox Palo Alto Research Center, to appear, 1980.
5. Ellis, Clarence, Gibbons, Robert, and Morris, Peter, "Office Streamlining," Institut de Recherche d'Informatique et d'Automatique, *Proceedings of the International Workshop of Integrated Offices*, Versailles, France, November 1979.
6. Cook, Carolyn L., "Streamlining Office Procedure," SSL-79-10, Xerox Palo Alto Research Center, in press.
7. Ellis, Clarence, Morris, Peter, and Smith, Stephen, "The Santa Clara Billing Office Study," Xerox ARG Technical Report No. 78-2, June 1978.



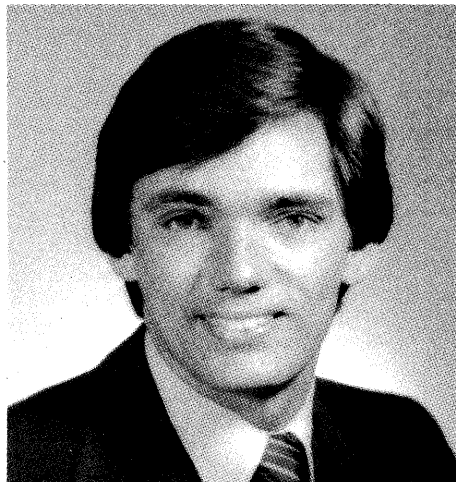
### **Office Automation in the Executive Suite: Successes and Strategies**

Real automation in the office will enhance, rather than replace, today's administrative and communication systems. The critical involvement of top-level executives as users of these new systems requires a "human approach" to design and implementation that challenges traditional assumptions of DP, MIS, WP and Telecommunications.

This session brings together three executives who personally use office automation to manage their organizations. They will describe the strategies which have enabled successful implementation of radical new forms of office communications and information systems and will

compare several approaches to the important issues of cost justification, information security, user interface design and inter-system compatibility.

Each speaker will describe how he uses office automation and the benefits his organization has experienced.



James Carlisle  
*Area Director*



# Provisions for flexibility in the Linköping office information system (LOIS)\*

by ERIK SANDEWALL, GÖRAN HEKTOR, ANDERS STRÖM, CLAES STRÖMBERG,  
OLA STRÖMFORS, HENRIK SÖRENSEN and JAAK URMI

Linköping University  
Linköping, Sweden

## 1. CHARACTERISTIC PROPERTIES OF THE LOIS SYSTEM

The Linköping Office Information System (LOIS) is an integrated system of facilities for text preparation, data base management, communication by computer, and miscellaneous other services. It is an experimental research system, which is used by researchers and secretaries in our own research group.

A significant consideration in the design of this system was how to provide very large flexibility, so that each user could have his or her customized variant of the system, without imposing an unrealistic burden of programming on either the users or a system group. Two complementary ways were recognized for achieving that flexibility:

- > *Adaptation by the user*: the system could include novel facilities which, like modelling clay, allow the user to adapt them to fit his/her needs;
- > *Application development tools*: there could be tools which enable a trained person to tailor facilities very easily, for individual users or groups of users.

Such tools should be easy to use, so that only moderate training is necessary, but there is no requirement that every user should be able to use them.

Both of these approaches have their merit, and both have been used in the LOIS system. The advantage of adaptation by the user should be clear; and application development tools are appropriate not only for harder tasks, but also for facilities which involve several users, i.e., what we shall call information-flow facilities below.

Another aspect of flexibility was that the standard services in the system should be easily interfaceable, so that they can be run together. This is a demand on the programming techniques that are used in building the LOIS system.

A second major purpose in building the system has been to experiment with unconventional terminal equipment. In particular, we have set up a low-cost device for output using

arbitrary fonts (boldface, italic, larger fonts for headlines, foreign alphabets, etc.) using a high-resolution electrostatic plotter, and built software for supporting that medium.

The following sections will describe these various aspects of the system in more detail. Thus the intended purpose of the paper is *not* to discuss the general-purpose facilities in office information systems. A number of significant and well-known system development efforts (for example at Stanford Research Institute, IBM Research Centers, and the University of Pennsylvania (Ref. 1)) have set the standard for such systems.

Before we proceed, we should however give a short summary of the services in the system, as seen from the individual user. LOIS recognizes three major ways of structuring information:

- text*, i.e. ordinary, continuous text in natural language (English, Swedish, etc.);
- data*, where information is organized as a table and/or as a form containing different fields or slots which may contain items of information;
- notes*, which is an intermediate form between text and data. A note is a short text which is associated with additional information organized as a *data* record. In practical usage, a note may be a message sent from one user to one or more other users, containing the text of the message plus information about sender, receiver(s), date, topic, etc. In another usage, the *note* may be one person's notes about the contents of a book, with associated information about author, title, and classification.

From another perspective, users will recognize some facilities as *local*, i.e., only one user is involved when they are used, for example for personal data bases, and other facilities which are *shared*, i.e., they involve the user in communication with other users, for example computer mail.

## 2. TEXT PROCESSING FACILITIES FOR FONTS

The LOIS system uses the common strategy of having general-purpose text editors and separate formatting ("run-

\* This research has been sponsored by the Swedish Board of Technical Development (STU) under contracts Dnr 77-4420, 77-4380b, and 78-4165.

off") programs. Besides supporting some conventional output media such as a Diablo printer, it also has a font printout system based on a Versatec electrostatic plotter. This system is able to produce output that approximates ordinary printing, with facilities for several *fonts*, such as italic font (cursive), **larger** and bold-face letters for major headings, etc. Different characters in a font may have different widths. It is possible to define fonts for other alphabets or for special signs, and use them freely mixed with the regular text. In particular, mathematical text as well as many kinds of figures may be produced in this fashion.

However, the graphic quality of this system is not fully commensurate with regular printing using typesetting. In particular, *italic* letters often appear a little unsteady if you look at them closely. We still believe that this quality is sufficient for many purposes. When compared to a phototypesetter, this equipment has the disadvantage of lower graphic quality, but also several advantages:

- the system is cheap enough that you can afford to have it within easy reach of each user;
- fonts may be created or modified at the site;
- the same device may also be used for vector plotting and grey-scale pictures (facsimile).

## 2. Text processing facilities for fonts.

The LOIS system uses the common strategy of having general-purpose text editors and separate formatting ("runoff") programs. Besides supporting some conventional output media such as a Diablo printer, it also has a font printout system based on a Versatec electrostatic plotter. This system is able to produce output that approximates ordinary printing, with facilities for several *fonts*, such as italic font (cursive), **larger** and bold-face letters for major headings, etc. Different characters in a font may have different width. It is possible to define fonts for other alphabets or for special signs, and use them freely mixed with the regular text. In particular, mathematical text as well as many kinds of figures may be produced in this fashion.

Figure 1—Sample printout from the Font System.

Technically, this system consists of a domestic LYS-16 16-bit small computer (soon to be replaced by an LSI-11 computer) combined with a Versatec graphic printer. The Versatec is an electrostatic raster printer with a resolution of 200 points per inch (8 points per millimeter). The LYS-16 contains software which will accept bit-pattern definitions of the characters in one or more fonts, and print a given file using these bit patterns for each character. The combined LYS-16 plus Versatec system may be viewed as an "intelligent printer."

The font system has been modeled on a similar system at the MIT and Stanford Artificial Intelligence Laboratories, which however use a Xerox Graphic Printer (XGP) instead of the Versatec printer. The resolution is almost exactly the same. On comparison, our system seems to give less contrast, but also less noise, and prints at a lower speed (probably mostly due to a slower processor).

### Formatting

The formatting for the font printout system is done using CRAWL, a locally built formatter which besides the support

of fonts, also has a number of other non-standard facilities:

- >automatic hyphenation (more necessary for Swedish text than for English text since Swedish makes frequent use of long, composite words—like German);
- >the formatter co-exists with a Lisp programming system, which means that commands in the source text can call arbitrary Lisp functions for specific purposes. This gives the same advantages as having a macro facility in the formatter (as is used in, e.g., the Unix system, Ref. 2), but with the significant difference that a full programming language provides services such as data base access, availability of a program library, and easy interface with other programs, which a macro system which only serves the formatter cannot be expected to provide.

A text formatter embedded in the programming language SAIL at the Stanford Artificial Intelligence Laboratory, offers similar advantages.

### Preparation of fonts

In the font printout system, each letter in each font is defined by a pattern of many small points. An ordinary small letter in a common font is about 15 points high, for example. The definitions of the point patterns of the characters in all fonts are stored on the central DEC-20 computer, and sent to the font printout device when needed.

The work of building up new fonts may require a considerable effort. Through the generosity of the M.I.T. A. I. group, we have a copy of their fairly large font library, which could be used after a routine shift of representation. However, we also have a need to modify old fonts (for example to create the Swedish letters with diacritics), and to create entirely new fonts for specific purposes.

Two tools have been built for these purposes, a *font editor* and a *font generator*.

### The font editor

The font editor is a tool for defining and changing the point-by-point definition of fonts. The font editor is in itself a program, but it requires a specialized terminal, which has been built by the Electrical Engineering department at our university. The system allows the user to edit one character at a time, and to view the character in two versions on the terminal's display screen, namely both a blown-up version where each point is clearly discernible, and a realistic version which looks like and gives the same impression as the character will have on paper (only magnified by about a factor of two). The editor allows the user to add and delete individual points or rows of points by hitting keys on the keyboard, and to see the effects of each change immediately.

The font editor has been very useful, both for modifying MIT fonts to contain Swedish characters, and for building up fonts of, e.g., mathematical symbols.

### The font generator

Although the font editor greatly facilitates the task of building up a font, doing so still requires a lot of work. Sometimes it is routine work, namely if letters of the same general shape are desired in several different versions, with different height, different boldness, roman or italic, with or without serif, etc. For such situations, we have developed a *font generator*, which generates fonts automatically from given specifications.

The font generating program takes two kinds of inputs. One input is the desired specification for the new font, i.e., values for the desired height of big and small letters, a measure of the desired boldness, etc. This input is specified anew each time the program is run.

The other input consists of structural *descriptions* of the characters in an alphabet, saying, e.g., that a capital "L" is a vertical line with a shorter horizontal line extending to the right from the base of the vertical line. These descriptions are expressed in a formal language, and are semi-constant, in the sense that the description of the Latin alphabet can be used repeatedly for different dimensions, but also if some other symbol set is desired (such as mathematical symbols) it is well defined how to write the structural descriptions of them also.

A program for the same purpose written by Knuth at Stanford uses mathematical functions (splines) to describe the curvature of the letters. Our system builds up letters from pre-defined segments, which can be designed by a combination of manual design and automatic generation. This is particularly useful for bit-matrix output devices whose resolution is almost discernible for the eye, since the effects of direct discretization of continuous functions may then be disturbing.

In addition, there are a number of smaller service programs for operating on fonts, such as a program for rotating the characters in a font by 90 degrees, and a program for rotating each page in a text file correspondingly.

### 3. STRUCTURED DATA FACILITIES

A significant part of the routines in an office environment deal with structured data rather than free text. The structured data facilities in the LOIS system, which aim to support this need, are organized around a screen-oriented *data editor* called IFORM. This system allows the user to view structured data on his display screen, organized into *forms*, i.e., fixed layouts containing certain fixed *text fields* and other fields, *data fields*, which can be filled with the desired data. Just like a text editor is used both for entering text and for changing existing texts, the IFORM data editor is used both for entering, viewing, and changing structured data.

Typical uses of the data editor in an office environment may be to maintain an address register, a register of reports and memoranda, a register of allocation of offices, or a register of equipment used in the group.

The basic idea in the IFORM system is of course available also in some commercial systems on the market, but IFORM

contains some facilities which are not usually found, in particular:

- >*programmability*: each data field may be associated with procedures in a number of different "slots" for defining specialized rules about how to interpret input into a field, check restrictions on the proposed input, print out the contents of a field on the screen, obtain consequences (side-effects) from new values, etc.
- >*tables within a form*: a form may contain a table which consists of a number of occurrences of a sub-record. This is useful for example when the form for a person contains a table of the trips he has made during the year, indicating the date, purpose, and destination of each, displayed with one line for each trip and one column for each field. Single-key editor commands allow manipulation of these sub-records, e.g., insertion and deletion of sub-records in the sequence.

To support this data handling facility, there are a number of other tools, in particular:

#### *Data base with exchangeable access methods*

The forms supported by IFORM are a standard interface for the user, through which he or she can access a number of different data bases, potentially even on several computers of different kinds. (This is in accordance with the proposals of the CODASYL End User Facility task group, and this idea has been articulated and extended within our *cjlaboratory* by Erland Jungert). IFORM is therefore organized so that access to the data base goes through a number of access routines associated with an *access method*. Additional access methods may relatively easily be added.

The ability to exchange access methods for the data base is in fact useful for two reasons:

- >for interfacing IFORM to a new data base;
- >for using one access method during development of an application and in its prototype stage, and another access method during production use of the same system.

#### *The layout editor*

IFORM uses a *form description*, i.e., a structure which describes the desired layout on the screen: which fields are used, what are their X-Y coordinates, etc. The *layout editor* is an interactive tool for building up and modifying such forms.

### 4. NOTES AND COMMUNICATION

The third information structure in the LOIS system, *notes*, are objects which consist of a short segment of free text, combined with a number of *properties*, each of which is a keyword and a corresponding value. The following is an



example of a note which a user may have during or right after a telephone call:

TALKWITH: Larsson

DATE: 1978-10-24

TOPIC: Holiday season, Vacations, Production

TEXT: Unusually many people are using remaining vacation days for extra vacation around Christmas. Production of bicycle chains will be particularly delayed.

The following is an example of a note which describes a computer terminal used in a research group:

TYPE: Hackmatic 1521

INVENTORY-NUMBER: 410

LOCATION: NB-156

CONNECTED-TO: DEC-20, PDP-11C

TEXT: This unit has required repeated service with various faults and seems to be flaky. Erasure of one line at a time does not work and seems to be permanently unfixable.

The POST subsystem in LOIS maintains for each user a database of notes, and enables the user to retrieve notes with given properties, to add new notes, to modify the properties of existing notes, and to call an ordinary text editor for modifying the textual content of a note. This information structure can be utilized for a number of different purposes, as suggested by the examples.

The POST system should really be viewed as a data base system which is able to also contain textual objects. It already provides non-trivial search facilities in this data base, and interfaces to other data base handling facilities, such as IFORM in the LOIS system, seems straightforward.

The present POST system encourages the texts to be short, but it is a straightforward extension to also allow notes whose text parts are conventional, larger text files for manuscripts. A system like POST might then be used as a more powerful substitute for the conventional file directory, and would allow the user to store arbitrary information about his files in the POST data base. This design would also give the user full data base capabilities for administering his "directory."

One particular use of notes is for *communication* between users, where each message is well expressed as a note, with properties indicating the names of sender(s) and receiver(s) of the message, the date the message was sent, the topic and other classification of the message, etc., and where the essential content of the message is conveyed in the free-text part, at least for simple messages. The POST system includes a message-passing facility, so that each user can send and receive messages, and the general-purpose data base facilities of POST can be used for administering incoming and outgoing mail.

Notice that the properties associated with the note are not only used for "system" purposes in the mail system, such as administrating the names of sender and receiver. They are also used by the sender and the receiver for representing information which classifies or otherwise describes the con-

tent, purpose, or use of the message. In particular, the receiver may change the values of properties, or add new properties, to messages that he has received. Also, it is sometimes very useful to represent some or all of the contents of the transferred message as values of properties, rather than in the free-text section.

One example of the use of such *structured messages* is the following: a message about a seminar may represent the name of the lecturer, the topic, the date, time, and location as separate properties. This greatly facilitates interfacing the message sending system to other facilities, such as a computer based calendar, or a system for generating summaries of recent activities.

The idea to base a computer mail facility on a data base handler for information organized as notes, appears to be a very powerful one. It provides a good basis for other communicative facilities, which may be more structured than simple mail sending, for example a *computer conferencing* system (which we have programmed but not yet put in operation), or for computer based decision making.

As the name indicates, the POST system started as a mail system, and its usefulness for storing one user's private information was recognized and exploited only gradually. The ability to organize one's personal information as a large collection of notes, and to have a full data base facility for keeping the notes organized, are only starting to be exploited, and we believe that several additional uses of this structure will be found as the system is used.

## 5. APPLICATION DEVELOPMENT TOOLS FOR INFORMATION-FLOW FACILITIES

The office environment contains many routines where a "packet" of information circulates between several "stations". For example, a purchase order is initiated by one person, and passes stations for approval, for selecting the vendor, for receiving and checking the goods, and for paying the bill. Each such application can be characterized as a flow of information packets, which follow certain paths; which sometimes are delayed awaiting some external event; which accumulate and give off information during their path through the organization; and which require human intervention at many of the stations.

As seen from the human user, these information flows are used for routine communication within the organization. In paper-based communication, one often prefers to use forms for this purpose, and in a computer-based system one would also desire fixed layouts (forms) rather than the free format of computer mail. For information flow with very high volume, for example in banks, this has of course been realized since a long time, but we are concerned with tools for low-volume information flow which must be supported locally.

Each information-flow application will involve several users, and symmetrically, one user will often be involved with several different information flows. In a hospital for example, the head nurse of a ward will be involved with at

least the following flows:

- >patient registers, undergoes treatment, and leaves;
- >"purchase" orders for laboratory analyses for patients in the ward;
- >scheduling of working hours for different categories of personnel in the ward;

and so forth. The entire office information system should therefore have a matrix structure with "users" in one dimension and "information-flow applications" in the other.

There is a significant structural difference between *development time* and *usage time*, then. When the system is used, each user wants to have his system as an entity, and to be able to switch easily between his part of each of the applications. In particular, he wants to be able to transfer data easily between the messages in different information flows. But when an application is developed, it is essential that all the work stations for that application are developed together.

Such information-flow applications are supported in the LOIS system by a combination of two measures. First, the software in the usage-time systems that are run by the individual users, have a well-defined structure so that additional facilities can be inserted automatically. Second, the LOIS system includes a modelling language and an application development tool which allow its user to build a description of an information-flow application in problem-oriented terms, and generate the appropriate contributions to the relevant usage-time system automatically.

The description of an application consists of three parts:

- >a description of the information flow as such, showing the successive operations (initialization, additional data entry, delay, copying, etc.) which happen along the way;
- >a record declaration which describes the structure of the information packets that travel in the flow;
- >a form description which defines the appearance of this record on screens and paper. This description is entered and maintained using the IFORM sub-system that was described above.

In addition, there is one master description of the organizational structure, which is used as a common reference by all information-flow models, and which relates them to the usage-time systems.

This application development tool is somewhat interesting from the point of view of programming methodology: usually a programming system handles entities ("programs") which contain the specification, or a part of the specification, for one executing process in the computer system. In our case, the application development system contains specifications for a set of coordinated processes, which are to be run by different users and often at different times, and which are all generated from the application description.

A more detailed description of this system has been given in Ref. 3. A system with some similarities has been developed by Hammer et al. (Ref. 4).

## 6. DIRECTORY SERVICES

Many parts of the LOIS system require that the system maintains directory information, i.e., information about information stored in the system. Examples of directory information are:

- >catalogues of the text files and data files maintained in the system;
- >classification information for notes;
- >structure descriptions ("declarations") for the data files maintained using IFORM, including information about the intended content of each data field.

In addition, there is directory information which is essential for the proper functioning of the system, but which is or at least should be invisible to the user, such as:

- >information about the different versions of a text file which appear in the course of successive operations (formatting, transcription to another alphabet, transformation to the printout conventions of a particular output device, etc.);
- >information about the access method used for a data file maintained by IFORM's data facility.

One basic design decision in LOIS has been that all such directory information should be maintained *in the data base of the system*, so that it can be accessed and used by the standard software facilities in the system, and by a gradually growing set of application programs. At present the following services are provided:

- >classification of data entities in an application-oriented hierarchical system, so that entities may be classified, e.g., with respect to what part of the owner's responsibilities they are used for. Such a structure is necessary when the number of text files and data files in the system increases: simple mnemonic naming of each file individually is not sufficient for structuring this body of information;
- >documentation of program modules, user systems, etc.;
- >automatically performing certain routine operations on text files, such as formatting and similar transformations before printout. This facility is viewed as a first step toward a system which "knows" about what routine data processing is needed in the application environment, and performs the appropriate operations at appropriate times. There are many similarities between this concept, and the modelling of information flow *between* users described in the previous section.

## 7. ARCHITECTURAL CONSIDERATIONS FOR FLEXIBILITY

New users are introduced to the LOIS system by learning about the basic facilities, for operating on texts, structured

data, and notes. But these sub-systems may be modified and recombined in many ways, and we expect that such modifications shall be done each time the system is used in a new environment or for a new class of tasks. It is not intended that every user should be able to modify the system, but it is intended that modifications can be done very close to the environment where they are going to be used, and preferably by one user of the system.

This flexibility of the system has been exercised to some extent within our environment, although additional experiments remain to be done. Several programming techniques are used to achieve flexibility and adaptability:

#### *Use of a residential programming system*

A residential programming system can be viewed as a data base system which is able to contain programs in its data base, and which contains an interpreter for programs that are stored there. Such systems provide unusual possibilities for program structuring, since programs and data can be integrated. This is useful for example for all programs that decode a repertoire of commands, and take appropriate action for each of the commands. There are many examples of such programs in office applications, for example editors and formatting programs for free text.

Another advantage of residential programming systems is that programs can be gradually modified and extended, even during an interactive session. This makes it easier to maintain a system as a collection of modules, which are loaded when needed.

#### *Rich parameter structures*

Several of the programs are directed by parameters which are represented as LISP list structures, which allows a rich and easily manipulated parameter language. Examples of use:

- >the IFORM data editor is parameterized with respect to layouts. The layout description specifies the location, content, etc., of each field. The non-trivial facilities in IFORM, such as for supporting embedded sequences of sub-records, depend strongly on this parameter structure;
- >the character description language used by the font generator, DRAW, is an example of a rich parameter language.

The use of a residential programming system facilitates the use of rich parameter structures, since programs and parameters are stored in an integrated fashion in the programming system's data base.

#### *Super routines, i.e., programs with handles*

Parameter structures are usually set up so that the parameters and/or the object data may contain the names of LISP

functions, which are called when the data are processed. This technique assumes of course equivalence between programs and data. Some examples of its use are:

- >the layout descriptions used by IFORM contain handles where calls to arbitrary (LISP) functions may be inserted, for specifying specialized printout formats, read-in functions, checking functions, or other aspects of the system's processing;
- >the POST sub-system allows messages and other notes to have a property which names a (LISP) function which is called when the note is processed. In this fashion it is possible to arrange that messages are processed automatically on reception, without need for manual intervention by the nominal receiver of the message. For example, a user may send out a query to a group of other users, where each query requires the recipient to answer a number of questions (represented as properties) and return the questionnaire, and where the initiator may set up a program which receives and summarizes the returns.
- >the CRAWL text formatter is designed so that the source file may contain calls to arbitrary (LISP) programs, which are executed when the call is encountered, and which, e.g., may generate a part of the desired printout (e.g., may make data base access and generate a table of structured data).

#### *Extendible command sets*

Several of the sub-systems contain specialized command languages, either for interactive use or for use in source files (in CRAWL). Usually they have been set up so that additional commands can be defined as LISP code in a modular fashion, and so that definitions for additional commands may be loaded into a sub-system even in the course of an interactive session. This technique makes it possible to keep the basic system small and simple. Instead of proliferating it with a large repertoire of special-purpose commands, the specialized commands are kept as separate modules and loaded into the system when needed.

- >the IFORM data editor may be extended with new commands which are specialized for application-oriented situations. For example, if IFORM is used to maintain information about patients in a hospital ward, one may have specific commands which are used when a patient enters or leaves the ward, and which initiate the operations (such as transfer of information to and from an archive) which are required at this event;
- >the layout editor which supports IFORM may similarly be extended with specialized commands, for example for introducing new kinds of fields. As one example, when the IFORM system was adapted to supporting VIEWDATA terminals, special commands were defined for inserting color shifts into the layout description.

### *Message passing between programming systems*

For each user, or group of users with similar needs, there is a version of the residential programming system which has been loaded with the programs, parameter structures, and other data which that user needs. Orthogonally to this set of *user systems*, there is also a set of *development systems*, namely one for each information-flow application, and one for each general facility (such as the formatter). The contributions which are made from development systems to user systems are transferred by a kind of message passing. The "systems" in this sense are therefore viewed as independent entities with local autonomy.

### *Combinability*

Another characteristic property of the system is that different modules can be made to interface with each other, using either "subroutine" calls or data transfer as the interface. This property of the system is made possible by a combination of two circumstances, namely (1) the flexibility properties which have just been described, and (2) the "callability" properties through all levels of software in the system we are using. This latter property is based on the TOPS-20 operating system, which for example makes it easy to let one process call another process recursively, including the operating system; but it is also due to the Interlisp system, which forwards these properties of the operating system to the programmer on the Lisp level.

The callability property has of course also been followed up within the LOIS system itself, where various sub-systems have been set up so that they can be operated both by direct user commands during an interaction, and as subroutines which are called from other programs.

Some examples of this combinability property in LOIS are:

- >the note handling system may call the text editors recursively, for operating on the textual content of a note. The same applies for the data editor;
- >the data editor has been equipped with a command which generates messages automatically using information in the data base, and calls POST for having them sent out to recipients. This is useful, e.g., for sending out reminders automatically according to criteria in the data base, such as a reminder to return a borrowed book when the time is out;
- >the text formatter CRAWL goes into a dialogue with the user when a syntax error in the input file is detected, allows the user to correct the error, and then proceeds through the same source file with no need to start over from the beginning of the file;
- >through the ability to define reception procedures for messages, it becomes possible to arrange that the contents of structured, incoming messages are gradually accumulated to the data base, where they can later be inspected using POST or IFORM

### *Additional services*

A few other programs have been written besides the basic facilities and their derivatives, in particular:

- >a personal calendar, with facilities for displaying and editing the current state of the calendar, and for booking a common meeting-time for several users of the system;
- >a personal agenda, i.e., a program which maintains a structured list of assignments that the user intends to perform, and provides support for editing this agenda.

## 8. IMPLEMENTATION TECHNIQUES AND EXPERIENCE

The LOIS system has throughout been intended as an experimental system, developed as a research project. The system has been designed so that it could be used within the group (for testing and for feedback on the design) but has not been intended for wider use. We therefore assigned high priority on the ability to modify and extend the system in the course of the project. For these reasons, and since we had access to a sufficiently large and powerful computer, we made the essential design decisions to let most part of the system operate on the DEC-20, and to write most parts of the system using the programming system INTERLISP. (Remaining parts have been written in assembler or Simula.)

At the same time, we also wanted to distribute some of the functions in the systems to separate and smaller processors. The locally built LYS-16 computer was used for this purpose.

In this final section, some aspects of this software strategy will be discussed.

### *Workspace systems vs. conventional systems*

Traditional computer programs operate with one or more files as input, similarly for output, and perhaps some interaction with a user. However, the INTERLISP system (like other LISP systems, and like APL systems) are organized so that the user will conduct an interactive session talking to a *system* which maintains a *workspace* for the duration of the session. This property is very significant for debugging and general maintenance of programs. It does not have to be used for the application situation, since one can write LISP functions which have the traditional file-in, file-out organization, but it is possible to use it for the application situation as well.

In LOIS, both approaches have been used. Some programs, such as DRAW and CRAWL, are essentially file-in, file-out, although with some possibilities for the user to initialize variables, etc., at the beginning of the session. Others, particularly POST and IFORM, rely heavily on LISP's workspace structure.

As a consequence, two different methods for maintaining structured data are both used:

- >a block of data (for example, one or a few "relations," or assignments of a number of "properties" to a number of "objects") may be stored as text files between sessions, and loaded into the data base when needed during a session. If data are changed, a new text file has to be produced, but this need only happen at the end of the session, or occasionally during the session but then only for reasons of backup and reliability. This method will be called *residential* storage of the data base;
- >alternatively, data elements (such as individual records in a relation, or property assignments to one "object") may be stored primarily as a segmented disk file even during the interaction session. Each data element is read into the workspace when it is needed, and if changed, the change is immediately performed on the disk file. This method will be called *external* storage of the data base.

Residential storage is the classical *modus operandi* in a LISP environment, and is very strongly supported by the INTERLISP system itself, which therefore is to be viewed, among other things, as a database system in the present context. External storage is sometimes advantageous, particularly when relatively little processing is performed on each data element, and when data and their updates are to be seen simultaneously by several users.

#### *Other useful properties of the INTERLISP system*

Some other properties of the INTERLISP programming system which were significant for the development of this system, are:

- >the very advanced support for program development activities: administration of programs, debugging, etc.;
- >the possibility to store parameter structures in the built-in data base (within the LISP workspace) and obtain services for the maintenance of this data base;
- >systems-programming facilities, such as easy interface to assembler code and to operating-system calls.

The major negative property of the system has been the relatively long time required to learn it. Since the language and the programming system is intended as a tool for the professional programmer, its high power must be paid by a relatively long learning time.

#### *Performance*

Since the intended purpose of the present project has been to develop an experimental system, which could be easily modified, but which also could be used within our group, the question of how much emphasis we should place on performance has recurred in the course of the project. Better performance can be achieved at the cost of more work and (often) a less transparent program. In particular, the use of LISP for major parts of the system represents a very high

priority for ease of development and maintenance, perhaps with a danger of slow performance.

Have we then obtained performance problems as a result of this strategy? This depends on how you look at it. Like most time-sharing systems in research environments, our computer system is sometimes badly overloaded, and the continued development work on parts of LOIS is not the least of reasons. However, if one judges the response times and general behavior of the LOIS system as seen by a user at times when the system is reasonably loaded (i.e., not thrashing), it seems that all major parts of the system are sufficiently quick for their intended purpose. The parts where response times are critical are the ones which have been programmed in assembler, and they form a relatively small part of the total software. The other parts, which have been written in LISP, are characterized either by a small amount of processing (although often of considerable complexity), especially in IFORM, or by a semi-batch mode of usage where longer execution times are tolerated especially if advantages of flexibility are offered instead.

This point may be illustrated with some figures. The CRAWL text formatter, entirely written in LISP, is about ten times as slow as the RUNOFF system, written in assembler. One should then remember that:

- >CRAWL provides certain additional services, such as variable-width fonts;
- >no attempt has been made to optimize CRAWL. A preliminary survey of what can be done indicates that there are several simple things one can do in the innermost loops, using short assembler routines;
- >the timings were made using the regular INTERLISP compiler; the block compiler could be used to speed it up.

In some cases, the first version of a program turned out to be too slow and had to be rewritten to gain speed. This only happened for a few, small programs (such as the low-level mail receiving program) and may to a large extent have been due to the programmer's short experience of LISP programming.

#### *Continued strategy*

In summary, we believe that the chosen implementation strategy has been a good one. Our continued strategy will be to develop additional facilities in LISP, and gradually improve the efficiency of existing facilities by a number of measures:

- >optimizing within the LISP context;
- >transfer by semi-automatic means to another programming language (for programs which do not need all of LISP's facilities);
- >transfer to smaller and cheaper processors for dedicated purposes, where CPU requirements may become less of an issue.

---

## ACKNOWLEDGMENTS

Many members of our group have helped with good ideas and constructive critique, in particular Jim Goodwin, Erland Jungert, John Walters, and Jerker Wilander, and Peter Fritzson and Dan Strömberg who also participated in the programming.

The variable-font printout system relies on several kinds of hardware built at the Electrical Engineering department of our university and in the Lysator society: the LYS-16 computer, the T2 special-purpose graphic terminal used for editing fonts, and others. In particular, we are grateful to Olov Fahlander for building the T2 and to Robert Forchheimer for a continuous interchange of ideas and information.

The project owes gratitude for the body of ideas and the software that we have inherited from the MIT Artificial

Intelligence Laboratory, from Xerox Palo Alto Research Center, and from Bolt, Beranek and Newman, Inc. in Cambridge, Mass.

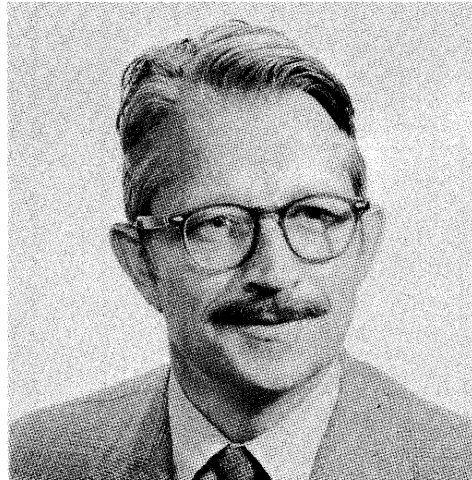
## REFERENCES

1. Morgan, H. L., *Office Automation Project*. Proceedings of the 1976 NCC Conference.
2. Kernighan, B. W. et al., *Unix Time-Sharing System: Document Preparation*. The Bell System Technical Journal, Vol. 57, No. 6, Part 2, July-August 1978.
3. Sandewall, E., *A Description Language and Pilot-System Executive for Information-Transport Systems*. Proceedings of the Fifth International Conference on Very Large Data Bases, Rio de Janeiro, 1979.
4. Hammer, M. et al., *A Very High Level Programming Language for Data Processing Applications*. Comm. of the ACM, Vol. 20, No. 11, Nov. 1977.



## **Security and Privacy of Data Flows**

The development of transnational computer-communication systems and the associated flows of computer data across international borders have created a number of issues and problems: privacy and security of personal data, non-tariff restrictions, concerns over potential erosion of national sovereignty, protectionism, and so forth. These developments are important to the data processing community in the United States since restrictions may be placed on the systems it develops and the data processing services it offers internationally. This session will address the issues involved in general, then concentrate on privacy protection problems, and finally explore a specific, new problem area—privacy rights of business, industry and other organizations regarding data about themselves.



**Rein Turn**  
*Area Director*





# Privacy protection and transborder data flows

by REIN TURN

California State University, Northridge  
Northridge, California

## INTRODUCTION

In the last decade there has been a dramatic increase in the growth of internationally operated computer-communication systems. In these systems, in essentially a single continuous operation, data are transmitted from terminals to computers in networks that may span several countries or several continents, the requested processing is performed, and results are returned. In other cases, data files are maintained on-line in international, remotely accessible networks. These networks are operated by vendors of remote computing and/or information services, industry associations, or private corporations (especially the so-called multinational corporations). Some of the data transmitted in these systems are personal data about individuals.

The world-wide availability of computer-communication and remote computing services has created a "trade" in these services, complete with competition between domestic and foreign vendors, taxes and duties, and regulations that appear to prefer domestic services. However, to date the data flows have been unbalanced. Raw data are flowing to very few highly industrialized countries where most of the international data processing service vendors and headquarters of multinational corporations are located. From these countries the processed data flow back to the originating countries. Many of the latter are the industrially less developed countries in the Third World. This situation has generated considerable concern in the originating countries over their excessive dependence on foreign data processing services, and over the lack of development or loss of business of their domestic data processing industry.<sup>1-4</sup> More detailed analyses of the underlying issues can be found in recent literature.<sup>5-7</sup> In general, a potential response in these countries may be to place restrictions on transborder data flows on the basis of the type and content of the data involved.

Privacy protection laws enacted in a number of European countries may provide one mechanism for restricting data flows. Privacy protection (also called "data protection" in Europe) emerged as a concern in early 1960s when automation of personal data record-keeping systems gained momentum. It was realized that automated systems are vulnerable to threats and subject to misuse on a scale that is significantly greater than in manually maintained record-

keeping systems, and that individuals should be provided with certain privacy rights—legally enforceable protection against unfair practices in collection, storage, processing, use, and dissemination of personal data about them. Beginning in early 1970s the United States, Canada, and seven European countries have enacted a variety of privacy protection laws,<sup>8</sup> and draft international agreements have been formulated.<sup>9-12</sup> These laws and agreements, and their impacts on transborder data flows, are discussed in the following sections.

## NATIONAL PRIVACY LAWS

Despite the differing perceptions of the problems and differing political and legal systems and traditions, the enacted privacy laws tend to grant individuals a remarkably similar set of privacy rights. A principal reason for this is that, from the beginning, privacy protection studies, discussions, debates and draft laws became widely known internationally. Thus, as the various countries tackled the problem and developed new concepts, others paid close attention and attempted to adopt these in ways that reflected their own situations. For example, the early developments (in 1969) in the Land Hessen of the Federal Republic of Germany,<sup>13</sup> the studies in Canada<sup>14</sup> and in United Kingdom,<sup>15</sup> and the Swedish Data Act of 1973 were widely studied.

Subsequently, a Code of Fair Information Practices was formulated in the United States<sup>16</sup> and international agreements on basic principles were reached in Council of Europe.<sup>9,10</sup> Later the Code was refined and expanded by the U.S. Privacy Protection Study Commission<sup>17</sup> and the international efforts in Europe<sup>11,12</sup> to include the following principles that are applicable to both the public sector (government) and the private sector's business, industry, and other organizations:

- *Openness*—there must be no personal data record-keeping systems whose very existence is secret, and there must be a policy of openness about any organization's record-keeping policies, practices, and systems.
- *Individual access*—there must be a way for individuals to find out what personal data about them are on record and how they are used, and to examine those data.

- *Individual participation*—there must be a way for individuals to correct or amend records of personal data about themselves.
- *Collection limitation*—there must be limits on the types of personal data that organizations may collect about individuals, and restrictions on the manner in which they collect these data.
- *Use limitation*—there must be a way for individuals to prevent personal data about themselves collected for one purpose from being used for other purposes without their knowledge or consent.
- *Disclosure limitation*—there must be limits on external disclosures of information about individuals which record-keeping organizations may make, and there must be legally enforceable confidentiality obligations of record-keeping organizations with respect to the use and disclosure of personal data.
- *Information management*—any record-keeping organization creating, maintaining, using or disseminating records of identifiable personal data must implement data management policies and practices which assure that the collection, maintenance, use, and dissemination of these data is necessary and lawful, that the data themselves are current and accurate, and that precautions are taken to prevent their misuse.
- *Accountability*—record-keeping organizations must be accountable for their personal data record-keeping policies, practices, and systems.

There are other dimensions of privacy protection, however, where there are considerable differences in national laws, especially between those in Europe and in the United States. Important dimensions are: the scope of applicability and coverage (government, private sector, both, or certain subsections of either); data subjects covered (individuals, legal persons, or both); types of systems covered (automated, manual, or both); and types of enforcement authorities and mechanisms. These aspects of the present privacy protection laws are briefly summarized below:

- *United States*. In the public sector, the Privacy Act of 1974 covers the automated and manual systems operated by the federal government, and protects the privacy of citizens and aliens admitted for permanent residence. Enforcement is through self-compliance and courts. The Office of Management and Budget has an oversight role. At states' level, privacy laws have been enacted in twelve states. They apply to automated and manual record-keeping systems under states' control, protect all residents, and are enforced in ways applicable to all laws in a given state. In the private sector, laws have been enacted to cover certain areas, such as credit reporting (Fair Credit Reporting Act of 1969), education (Family Education Rights and Privacy Act of 1974), and financial institutions (Right of Financial Privacy Act of 1978). These cover manual and automated systems, all individuals, and are enforced by agencies traditionally assigned legal oversight roles in these areas. Bills are pending in Congress to expand privacy protection to other areas of the private sector, such as employment records, health care, and insurance industry.
- *Sweden*. The Data Act (1973, amended in 1977) covers automated record-keeping systems in both the public and the private sectors. Covered are all residents. Enforcement is through the Data Inspection Board. Permission is required to export personal data. Other provisions of the Act reflect the record-keeping environment in Sweden: most of the record-keeping systems are automated, each citizen is assigned a unique personal identification number that can be used to link various records on an individual into a complete dossier, and Sweden is a very open society where, for example, the income and taxes of the citizens are published by the government for open distribution.
- *Federal Republic of Germany*. The Federal Data Protection Act (1977) covers automated and certain manual record-keeping systems in the public and private sectors. All residents are protected. Law is enforced by the Federal Commissioner for Data Protection. There are provisions for limiting disclosure to foreign organizations.
- *Canada*. A section of the Canadian Human Rights Act (1977) applies to automated and manual systems of the federal government. Protected are citizens and aliens admitted for permanent residence. Enforcement is effected through the office of the Privacy Commissioner. Certain TDF cases require Commissioner's approval. In the provinces' level, fair information practices laws are in force in several provinces. In the private sector, privacy protection is applied in certain areas (such as credit reporting).
- *France*. The Act on Data Processing, Data Files and Individual Liberties (1978) covers automated systems and certain manual files in the public and the private sectors. All residents are covered. Enforcement is by the National Data Processing and Liberties Commission. Permission is required for transborder transfers of personal data.
- *Norway*. The Act Relating to Personal Data Registers (1978) covers automated systems in both the public and the private sectors. Protection is provided to individuals, and to associations or foundations. The law is enforced by the Data Surveillance Service. Permission is required for transmission of personal data abroad.
- *Denmark*. The Public Authorities' Registers Act (1978) covers government agencies that maintain automated records on residents. It is enforced by the Data Surveillance Authority. License is required for TDF. In the private sector, the Private Registers Etc. Act (1978) applies to automated systems in the private sector and protects individuals and institutions, associations and business enterprises. It is enforced by the Data Surveillance Authority. No TDF provisions are made in this law.
- *Austria*. The Federal Act on the Protection of Personal

Data (1978) covers automated record-keeping systems in both public and private sectors. It protects individuals and legal persons or associations. Enforced by Data Protection Commission and Council. TDF provisions include a requirement to obtain permission to export data, and to process data on foreign persons in Austria.

- *Luxembourg*. The Law Governing the Use of Name-Linked Data in Data Processing (1978) covers automated systems in public and private sectors, and protects individuals and legal persons. It is enforced by existing governmental authorities. In TDF situations, if the data access point is in Luxembourg, the law applies.

In addition, privacy protection laws are pending in Belgium, the Netherlands, and Portugal. Privacy protection requirements have been incorporated in the constitutions in Austria, Belgium, Portugal, and Spain. Still other countries are in study phases that are expected to produce privacy protection legislation in the near future (e.g., Finland, Japan, Switzerland, and United Kingdom). It is important to note the tendency in the more recent national privacy laws (Norway, Denmark, Austria, and Luxembourg) to extend privacy protection to legal persons. That is, corporations, associations, and other organizations are granted the same rights regarding data about them as are given to individuals. Implications of these extensions are far-reaching.<sup>18</sup> For example, if transborder data flows were restricted under privacy protection provisions of national laws, much greater proportion of data flows would be covered if legal persons were included.

## INTERNATIONAL HARMONIZATION

The similarities and the differences among national privacy protection laws can become important practical matters to governments and organizations in the private sector when transborder data flows are being considered. On one hand there is the problem of comparisons of various features in national laws in order to determine which laws are "stronger," on the other hand there is the problem to participants in transborder data flows of complying with different implementations of the same requirements. Thus, there is a general agreement that it is desirable to standardize and "harmonize" the basic privacy protection provisions in the laws of various communities of nations. In response to this, two organizations in Europe have produced draft documents—the Council of Europe (located in Strasbourg, France) and the Organization of Economic Cooperation and Development (located in Paris, France). The former is an organization of 21 Western European countries. OECD also includes non-European countries, such as the United States, Japan, Canada, Australia, and New Zealand.

The Council of Europe has drafted a Convention for the Protection of Individuals with Regard to Automatic Pro-

cessing of Personal Data<sup>11</sup> in order to establish the following:

- A minimum set of privacy protection principles and rights to be adopted by all signatory countries (countries that are not members of the Council of Europe will be invited to join).
- Obligation of signatory countries to grant basic privacy rights to all individuals, regardless of nationality or residence.
- Cooperation and exchange of information between national data protection authorities in supervising compliance with privacy protection laws in international settings.
- Administrative mechanisms to handle disputes over jurisdiction when implementing national privacy protection requirements to handle TDF situations.

The Convention would commit the signatory countries to enact privacy protection laws based on the principles listed in the previous section. It provides a privacy protection "floor" acceptable to the signatory countries in the sense that they would consider any country that has enacted and is enforcing these principles to be providing "sufficient" privacy protection. Then personal data could be transmitted to such a country from other signatory countries without loss of basic privacy protection. The Convention is to cover automated record-keeping systems in public and private sectors. Protection is afforded to individuals, but could be extended to cover also manual systems and/or legal persons by any signatory country. However, presumably, the lack of such extensions in a signatory country's privacy laws would not be considered a sufficient reason for restricting transborder data flows to this country by signatory countries that do have these provisions.

A parallel effort toward standardization and harmonization of privacy protection principles, rights, and requirements in the form of a set of Guidelines Governing the Protection of Transborder Data Flow of Personal Data<sup>12</sup> has been completed by the OECD. The Guidelines are to be voluntary (not a legally binding treaty) on the OECD member countries that accept them, but they are regarded as "morally bound" to comply fully. The United States is expected to participate.

The OECD Guidelines are based on the philosophy that there are economic and social benefits to all participants in transborder data flows, and that it is very undesirable to establish unjustified barriers to TDF. Their purpose is to provide an interim standard until more formal treaties such as the Council of Europe's Convention are adopted. Again, the expectation is that there would be no need to restrict personal data flows between countries that have accepted and are implementing the Guidelines' principles and requirements. For this purpose, the Guidelines are similar to the Council of Europe's draft Convention. That is, they are designed to be applicable to both the public and the private sectors, and to manual as well as automated systems. The protected data subjects are defined to be individuals (physical persons).

Regarding transborder data flows, the Guidelines urge the participating countries to observe the following:

- Take into account the implications on other signatory countries of domestic processing and re-export of personal data, particularly when this may result in circumvention or violation of national privacy laws of other signatory countries.
- Take all reasonable and appropriate steps to ensure that transborder flows of personal data, including transit-only data, are uninterrupted and secure.
- Refrain from restricting transborder data flows of personal data between themselves and other countries except when these do not yet substantially observe the Guidelines.
- Refrain from developing laws, policies and practices in the name of the protection of privacy and individual liberties which, by exceeding requirements for these, are inconsistent with free transborder flow of personal data.
- Insure that their procedures for TDF of personal data and for protection of individual liberties are simple and compatible with those of other signatory countries.
- Work toward development of principles, national and international, to govern the determination of applicable laws in the case of TDF of personal data.

In addition to the Guidelines which are expected to be ready for adoption relatively soon (e.g., in 1980), OECD will examine the question of restrictions on TDF of data not related to individuals, such as data on legal persons, national economy and resources, and so forth. Further, to address the non-privacy issues in TDF briefly mentioned in the Introduction, the OECD is proposing that its member countries consider adopting a "Transborder Data Flow Pledge" for a limited time period, affirming their agreement to:

- Avoid adopting national policies of restricting imports of data processing services or taking any actions to restrict TDF between the participating countries;
- Refrain from discriminating against imported data processing or telecommunications services;
- Avoid taking measures to place supplemental taxes on imported information and data processing services.
- Cooperate in setting up an international legal framework covering various aspects of TDF;
- Consult with each other on implementation of the pledge in accordance with international obligations, and taking into account the special needs of developing countries.

At this writing, none of the above international initiatives have been adopted by respective policy making bodies. There is little question, however, that they would be very useful in setting up a proper framework for handling the numerous TDF issues.

## IMPLICATIONS ON TRANSBORDER DATA FLOWS

Businesses and industry in the United States, especially the multinational corporations and vendors of remote computing and information services, have a strong interest and dependence in unrestricted transborder data flows. Until the international agreements have been firmed and accepted by a significantly large number of countries, including the United States, the national privacy protection laws of the European countries could be used to justify placing restrictions on data flows from these countries to the United States.

Privacy protection in the context of transborder data flows deals with extending any privacy rights that individuals may have in their "home country" to any "host country" where personal data about them may be processed, stored or used. The purpose is to assure that:

- Individuals could continue to exercise their home country privacy rights regardless of the physical location of the personal data about them.
- Record-keeping organizations in the home country could not export personal data abroad in order to evade privacy protection requirements in the home country. That is, they should not be able to establish data havens abroad, nor make use of any existing ones.
- Personal data maintained abroad would be protected from unauthorized use or dissemination by parties in the host country, and from deterioration of quality (accuracy, completeness, currency) while abroad or in transit.
- Individuals would have privacy rights vis-a-vis personal data about them collected directly in the home country by foreign organizations, or collected while they are abroad.

In general, reaching these goals requires that privacy protection laws in the host country provide protection to foreign nationals, special agreements exist to permit foreign nationals to exercise their privacy rights in host countries (e.g., the international initiatives discussed in the previous section), and contracts or licenses bind involved organizations in the host country to abide by data confidentiality, security and quality requirements. If agreements cannot be made to the satisfaction of a home country's privacy protection authorities, these may apply the TDF clauses in their privacy laws to restrict personal data transmissions to countries that have weaker privacy protection laws.

The results of comparisons of the privacy protection afforded in the United States versus protection afforded in Europe as based strictly on privacy protection laws now in force are likely to indicate that privacy protection in the United States is less comprehensive in the following ways:

- The scope of coverage is narrower since both the public and private sectors are incompletely covered in the United States — even though the federal government is covered by the Privacy Act of 1974, only twelve states

have enacted fair information practices laws. In the private sector, only consumer credit, educational institutions, and financial institutions are covered. European privacy laws cover both sectors quite completely.

- The Privacy Act provides privacy protection explicitly only to citizens of the United States and aliens admitted for permanent residence. Thus, foreign nationals other than specified above are not covered. European privacy protection laws appear to make no distinction regarding nationality or citizenship.
- No central, independent privacy protection authority exists in the United States to enforce compliance with privacy protection laws. Under the Privacy Act, Office of Management and Budget has a nominal role of coordinating compliance, and the President makes an annual report on compliance to the Congress. Compliance with federal privacy laws that apply to the private sector is with agencies that normally regulate the areas involved (e.g., the Federal Trade Commission, the Federal Reserve System, and the Department of Education). This distributed authority contrasts with the strong, central data protection authorities in Europe.

However, comparisons of privacy protection laws are likely to give a misleading picture of the general level of privacy protection in the United States. The scope of the U.S. privacy laws is broader since they cover also manual record-keeping systems. Private sector privacy laws apply to all residents. The state fair information practices laws are only a small fraction of privacy protection activities in states. More generally, the Constitution of the United States, and those of the individual states, place strong emphasis on openness in governmental decision making processes, and establish an atmosphere of concern for individual rights. The lack of a central privacy protection authority is, itself, an illustration of the American aversion to concentrating power in government agencies. Practices such as a universal identification number for each citizen or publishing the earnings and income of all citizens, as presently exist in Sweden, would not be acceptable to Americans. Finally, to reduce the incompleteness of coverage in the private sector, bills were introduced in 1979 to establish fair information practices and place limits on use of personal information in the maintenance of medical records, financial information, insurance records, and research records.

The differences in scope of coverage of the national privacy protection laws have resulted from differing philosophies of regulation. The European countries have adopted an "omnibus" approach that applies the same set of requirements (usually with some exceptions) uniformly to the government and all parts of the private sector. In United States and Canada, where federal system of government is strongly established, based on considerable separations in powers and jurisdiction, a sectoral approach has been taken — separate laws are enacted (or existing laws are amended) to provide privacy protection in specific parts of the public and private sectors. Each approach has its merits and draw-

backs. Omnibus legislation establishes uniform requirements, but cannot handle easily any specific situations. Sectoral legislation is easier to enact and permits flexibility in handling exceptions, but is likely to result in scattering of privacy protection requirements throughout the entire legal code.

In view of the above discussion, it is important for the U.S. enterprises involved in TDF and desiring to continue, to find means to convince the privacy protection authorities in the countries they operate of their commitment to privacy protection principles, and their willingness and ability to provide privacy protection at levels that match the requirements in the respective countries. The specific measures taken depend on the nature of the enterprise and its TDF activities, the nature of data involved and processing performed, any uses made of the data by the organization, and the laws in force in the home country. An effective approach appears to be the voluntary compliance and implementation of privacy protection requirements that apply to the private sector area involved (e.g., adoption of the recommendations of the U.S. Privacy Protection Study Commission that apply to the area<sup>17</sup>), as well as adoption of the principles of the OECD Guidelines.

Voluntary compliance has been urged by the Privacy Protection Study Commission, by the President of the United States at the introduction of his privacy protection initiatives in the Spring of 1979 (with special emphasis on voluntary compliance regarding employment and commercial credit granting records), and by industry and business groups, such as the Chamber of Commerce of the United States<sup>19</sup> and the Business Roundtable.<sup>20</sup> In addition, codes of conduct or ethics have been suggested as one avenue toward effective voluntary compliance with privacy protection principles and requirements in national as well as TDF contexts. On an industry association level, codes of conduct and effective sanctions for noncompliance could discourage organizations from using personal data contrary to privacy protection principles. On the employee level, such codes could discourage improper handling of personal data, especially the personal data on foreign nationals. However, at this time codes of conduct or ethics are not sufficiently strong to be depended upon for satisfying privacy protection requirements.

Compliance with privacy protection requirements can involve substantial changes<sup>21</sup> in operating procedures, record-keeping systems and practices, personnel requirements and training, and substantial costs.<sup>22</sup> Thus, there are few incentives for voluntary compliance without any other compelling reasons, such as continued participation in TDF activities. While a number of large firms have adopted voluntary privacy protection programs (usually regarding employment records) others appear to be delaying. For example, a recent survey of the top 500 corporations in the United States (with 145 respondents) showed that most of the respondents have not informed their employees of dissemination of their records to credit agencies, and over 80 percent do not allow their employees access to medical information about them used in employment-related decisions.<sup>12</sup> Some foreign privacy protection authorities have already cited these results

as evidence that there is little intent by U.S. enterprises to abide by voluntary guidelines proposed by OECD.

Despite the present difficulties in understanding how voluntary compliance could be achieved to satisfy privacy protection authorities in other countries that the privacy rights of their nationals will not be reduced when their personal data are in the United States, voluntary compliance is an alternative that could be implemented quickly and in ways that adapt readily to the specific requirements in any particular transborder data flow case. Efforts must be continued to develop an effective approach and implementation mechanisms.

### CONCLUDING REMARKS

Transborder data flows among certain industrial countries are increasing, but most of the data processing is done in only a few countries, especially in the United States. This is viewed by countries where data flows originate as not in their national interests. At the same time, concerns over human rights have led to the enactment of privacy protection laws in numerous countries in Europe, and in the United States, with many other countries likely to follow. At the present time, the enacted or proposed privacy laws in Europe are more comprehensive than U.S. laws, and can provide a rationale for restricting transmissions of personal data to the United States. One mechanism that U.S. enterprises involved in transborder data flows can use to reduce the potential for restrictions is to adopt the OECD guidelines voluntarily until governmental or legislative initiatives in the United States have reduced the present discrepancies.

### ACKNOWLEDGMENTS

This paper is based, in part, on the report by AFIPS Panel on Transborder Data Flows<sup>5,8</sup> which the author edited as chairperson of the Panel. Major contributors to the report were panel members Roger N. Allen, George I. Davida, Eric J. Novotny, Susan H. Nycum, David M. Rappaport, Alexander D. Roth, Phillip A. Tenkhoff, and Willis H. Ware.

### REFERENCES

1. *The Vulnerability of Computerized Society*, Ministry of Defense, Stockholm, Sweden, 1978.
2. Clyne, J. V. (Chrmn.), *Telecommunications and Canada*, Consultative Committee on the Implications of Telecommunications for Canadian Sovereignty, Ottawa, March 1979.
3. Nora, S. and A. Minc, *Report on the Computerization of Society*, Board of Financial Examiners, Paris, France, 1976.
4. "SPIN Conference Resolutions," *IBI Newsletter*, No. 27, IBI, Rome, Italy, 1978.
5. Turn, R. (Chrmn.), *Transborder Data Flows: Concerns in Privacy Protection and Free Flow of Information, Vol. 1*, AFIPS Panel on Transborder Data Flow, Washington, DC, December 1979.
6. Allen, R. N., "Overview of Transborder Data Flow Issues," Computer Sciences Corporation, El Segundo, CA, November 1979.
7. Langhorne, R. W., "Private Enterprise Concerns About Data Protection and Transborder Data Regulation," *Data Regulation: European and Third World Realities*, Online Conferences, Ltd., Uxbridge, England, November 1978, pp. 135-158.
8. Turn, R. (Chrmn.), *Transborder Data Flows, Vol. 2., Supporting Materials*, AFIPS Panel on Transborder Data Flow, Washington, D.C., December 1979.
9. *On the Protection of Privacy of Individuals Vis-A-Vis Electronic Data Banks in the Private Sector*, Resolution (73)22, Council of Europe, Strasbourg, France, 1973.
10. *On the Protection of Privacy of Individuals Vis-A-Vis Electronic Data Banks in the Public Sector*, Resolution (74)29, Council of Europe, Strasbourg, France, 1974.
11. *Draft Convention for the Protection of Individuals With Regard to Automatic Processing of Personal Data*, Council of Europe, Strasbourg, France, May 1979.
12. *Draft Guidelines Governing Protection of Privacy and Transborder Flows of Personal Data*, OECD, Paris, France, June 1979.
13. Hondius, F. W., *Emerging Data Protection in Europe*, North-Holland Publishing Co., Amsterdam, 1975.
14. *Privacy and Computers*, Department of Communications/Department of Justice, Information Canada, Ottawa, Canada, 1972.
15. Younger, K. (Chrmn.), *Report of the Committee on Privacy*, Cmnd. 5012, Her Majesty's Stationery Office, London, 1972.
16. Ware, W. H. (Chrmn.), *Records, Computers, and the Rights of Citizens*, Secretary's Advisory Committee on Automated Personal Data Systems, Department of Health, Education and Welfare, Washington, D.C., July 1973.
17. Linowes, D. F. (Chrmn.), *Personal Privacy in an Information Society*, Privacy Protection Study Commission, Washington, D.C., July 1977.
18. Nycum, S. H., "Issues in Legal Person Privacy," *AFIPS Conference Proceedings, Volume 49, 1980 NCC*, AFIPS Press, Washington, D.C., 1980 (this volume).
19. *Personal Information Privacy*, U. S. Chamber of Commerce, Washington, D.C., June 23, 1978.
20. *Fair Information Practices — A Time for Action*, Business Roundtable, New York, December 1978.
21. Turn, R., "Privacy and Security in Transnational Data Processing Systems," *AFIPS Conference Proceedings, Vol. 48, 1979 NCC*, AFIPS Press, Montvale, NJ, June 1979, pp. 283-291.
22. Turn, R., "Privacy Protection Costs in Record-Keeping Systems," *Information and Privacy*, September 1979, pp. 298-302.

# Transborder data flow: legal persons in privacy protection legislation

by SUSAN H. NYCUM and SUSAN COURTNEY-SAUNDERS

*Chickering & Gregory*  
San Francisco, California

## SYNOPSIS

In view of current proposals to establish treaties and privacy protection guidelines on transborder data flows, it has become necessary that the United States adopt a position on whether "legal person" be included in such treaties and guidelines. The following discussion attempts to evaluate the various arguments and concludes that the United States, based upon its own definition of privacy, regulation of "legal persons," and economic and foreign policy, must maintain that "legal person" be excluded from privacy guidelines.

## DISCUSSION

### A. Introduction

Inclusion of "legal persons" in regulation of privacy with regard to transborder data flow encompasses sociological, political, economic and philosophical issues. Review of the issue of protection of individual privacy is not enough. While certain nations seek control of transborder data flow through access to its use, process and content, other nations desire to foster an atmosphere of free enterprise and competition by permitting business to remain relatively free of governmental or private intrusion.

Including "legal persons," or natural persons and business entities, in privacy protection may inhibit the free flow of data and free exercise of competition. If business entities are included in privacy protection, any business files would be open to inspection and correction by any individual or other business mentioned in those files. As a result, customers, potential customers, competitors, and suppliers would have the opportunity to inspect the files of the particular business. Such disclosure may be so harmful to business that it cannot function effectively in that country.<sup>1</sup>

On the other hand, privacy rights may actually be further protected if business entities are entitled to inspect data concerning them. This may be the case when an individual's privacy interest is so identified with a business entity's interests that the individual's privacy can be more fully protected through disclosure of business records to him. Also,

certain countries have political, economic or social reasons for including legal persons in their privacy legislation, not the least of which is to exclude competition presented by the foreign data industry.

The United States must consider its own policies on privacy regulation and international commerce and other countries' concerns in order to protect its own individual, corporate and national interests and to harmonize them with interests of other nations.

This paper will first outline current definitions of "legal persons," existing privacy protection legislation covering transborder data flow (hereinafter TDF), and current proposals for treaties and guidelines on privacy regulations and TDF.

Next, arguments for and against inclusion of "legal persons" in privacy protection will be presented.

Finally, the considerations the United States must address on this issue, and the recommendations it might make, will be discussed.

### B. Existing and proposed privacy protection in TDF regulation

#### 1. Definition of "legal persons"

In order to examine the inclusion of "legal persons" in TDF privacy protection, a single definition must be agreed upon.

Under the Internal Revenue Code, a "person" is an individual, trust, estate, partnership, association, company or corporation.<sup>2</sup> Under the New Bankruptcy Act, "person" includes an individual, partnership or corporation.<sup>3</sup> Although the Freedom of Information Act defines "person" as including "an individual, partnership, corporation, association, or public or private organization other than an agency,"<sup>4</sup> the Privacy Act limits its protection to "individuals" or "a citizen of the United States or an alien lawfully admitted for permanent residence."<sup>5</sup> The Right to Financial Privacy Act provides access to financial records by any "individual or a partnership of five or fewer individuals."<sup>6</sup>

Authorities on TDF regulation have stated that a "legal



person" does not have to be a legal entity, but can be any group of persons<sup>7</sup> or any business enterprise.<sup>8</sup>

If "legal persons" are to be included in the group protected by TDF privacy regulation, then the definition of "persons," based on the above authorities, would probably be natural persons, groups of persons, and any business entity including a sole proprietorship, corporation or partnership. For purposes of this report, "legal persons" will signify all the above entities *except* natural persons in order to distinguish regulation solely of natural persons from regulation of both natural and business persons.

## 2. Current privacy protection legislation

### a. Sweden: the Swedish data bank statute of May 11, 1973 (1973:289)

The Swedish Data Bank Statute is the earliest regulation of TDF and privacy. It requires all domestic and foreign data banks containing personal information to be licensed by the Data Protection Board before they begin operation in or with Sweden. A registration list containing the information stored or transmitted on persons must be given to the Board. The protection extends only to private persons, who may request from the Board any information in the list concerning them. Section 1, 16. Thus, Sweden does *not* include legal persons in its privacy legislation.

### b. West German republic: federal data protection act, 1977

This Act provides for a private data administrator at each business storing data on individuals. This administrator decides what categories of privileged information are to be stored and what processing to use. Section 29. Also the Act provides for disclosure by the government of data held on individuals. Sections 7-21. The German Act covers only physical persons. Section 2(1).

### c. Canada: human rights act, 1977, part IV: protection of personal information, house of commons, 2nd session, 30th parliament, 25 Eliz. II, bill C-25

This bill provides only individuals, and not legal persons, with the right to obtain, examine and correct records concerning them held by or for the Canadian Government. Part IV, Section 49, 52.

### d. France: act no. 78-17 of 6 January 1978 concerning computerized indexes and the protection of the liberties

The French Act requires that any entity desiring to establish a data bank must obtain an authorization from the Na-

tional Commission for Data Processing and the Liberties. Chapter III, Art. 14. Also, any data bank must be open for inspection and correction only to individuals. Chapter I, Art. 4; Chapter V.

### e. Norway: personal data registers act of 1978

This Act requires that data banks obtain permission from the King in order to establish personal data registers. Chapter 4, Section 9. The "personal information" covered by the legislation includes "information . . . traceable to identifiable individuals, associations or foundations." Chapter I, Section 1. Thus, those Norwegian "persons" protected include *both* individuals and business entities.

### f. Denmark: bill on private registers 1978

The Danish Bill requires data banks to register with the Register Board prior to communication to any third party in Denmark. Chapter 3, section 14. Legal persons are included in this privacy protection because "[a]ny systematic collection and registration of information on the financial conditions of persons, *associations or undertakings*" may take place only in accordance with the act. Chapter 1, Section 1. The act provides that any "person" may inspect any information about themselves held by any registered entity. Chapter 2, Section 7.

### g. Austria: act on data protection, 1978

The Austrian Act requires private data banks to register with and obtain a license from the Federal Minister, including data transmitted abroad unless the receiving country has a data protection law similar to the Austrian law. Chapter II, Section 9; Chapter V, Section 25. The act includes legal persons by defining "personal data" as "information, including personal identification marks, relating to a natural person or a *juristic person* or a '*personal company*' in the sense of commercial law." Chapter I, Section 2.

### h. Luxembourg: act regulating personal data use in data processing, March 1979

This Bill requires any data bank using a Luxembourg data processing means and containing personal data to obtain authorization by a competent minister and registration with a general directory. Chapter 2, Section 4,5. The Bill defines persons as "any natural person . . . [e]ach corporate body under Luxembourg law, from time of constitution, (or) any other corporate body registered in a Luxembourg Public Administration or a Social Security establishment register." Chapter 1, Art 2. All "persons" have the right to inspect and correct data contained in any data bank registered with the National Directory. Chapter 5, Art. 19,20.

i. United States of America: freedom of information act, 5 U.S.C. §552 (1974) and the privacy act 5, U.S.C. §552a. (1974)

The Freedom of Information Act (FOIA) provides the mechanism for any "person," including an individual, partnership, corporation, association, or public or private organization, to be provided, upon request, with any non-exempt record held by a government agency about any "person,"<sup>9</sup> except information that would, if disclosed, violate national security, trade secrets or other confidential business information, attorney-client privilege, or, balancing the interest served by the exemption (e.g., privacy) against the public interest served by disclosure, information on individuals may be withheld. 5 U.S.C. §552(b)(1-7).

The Privacy Act permits any "individual" or citizen of the United States or an alien lawfully admitted for permanent residence to inspect records only when held by any government agency. It also prohibits disclosure by a government agency of any record of an individual without the written consent of the individual if the record constitutes a clearly unwarranted invasion of personal privacy, such as medical files. The FOIA's balancing to determine the exemption from disclosure for individual data is not present in the Privacy Act, which provides for a clearer restriction on disclosure. 5 U.S.C. §552a(b), §552(b)(6).

Neither act provides for disclosure of information by *private entities* to either natural or legal persons. Bills are pending in the Congress which would extend such coverage to the private sector.

Privacy protection extends to providing the right to examine records held by private entities, however, under the Right to Financial Privacy Act, 12 U.S.C. §401, *et seq.*, which permits individuals or partnerships of five or less to inspect information on them held by any public or private financial institution, and the Fair Credit Reporting Act, 15 U.S.C. §1681, *et seq.*, which permits individual consumers to inspect credit reports held by a consumer reporting agency.

### 3. Current and proposed TDF international treaties and guidelines

#### a. The council of Europe

The 20-member Council of Europe (COE) expects to ratify its treaty on data protection, including privacy protection, by 1980. The U.S. will have the opportunity to be a signatory along with Japan, Canada and Australia. Whether the U.S. signs the treaty or not, it will be affected.<sup>10</sup>

The purpose of the COE treaty is to establish common rules among nations on the rights of the individual when an automated record on the individual is gathered, processed, and transmitted across borders. The treaty must consider and reconcile the issues involved: protection of the rights of the individual, preservation of the free flow of informa-

tion, and recognition of the supremacy of national law and variations in legal systems.<sup>11</sup> It will represent only minimum rules and each nation can expand the scope of protection.<sup>12</sup>

The current treaty draft does not cover data on legal persons.<sup>13</sup> However, the drafters are trying to provide a mechanism for cooperation between those nations including legal persons and those excluding them. Frits Hondius, Division head of COE in Strasbourg, France, indicates that the "legal persons" issue has not yet been resolved by the COE: "Where a legal person should be able to claim the same protection as a natural data subject is a controversial question that needs careful study."<sup>14</sup>

#### b. The OECD

The U.S. is participating in the drafting of guidelines for the cooperation on TDF issues by the Organization for Economic Cooperation and Development (OECD).

The OECD has omitted from its third draft a section providing that its guidelines would not restrict any nation from applying privacy protection beyond natural persons "to groups of persons, associations, corporations or any other bodies whether or not such bodies possess legal personality."<sup>15</sup> Its current intention is therefore to exclude "legal persons" from privacy protection, replacing that clause with:

"3. These Guidelines should not be interpreted as preventing the application of different protective measures to different categories of personal data depending upon their nature and the context in which they are collected, stored, used or disclosed. *Furthermore, nothing in these Guidelines should be interpreted as preventing countries from applying them only to the automatic processing of personal data.*"<sup>16</sup>

#### c. OECD vs. COE: the need for uniformity

While the COE concentrates on the issue of protecting individual rights, the OECD is looking at the total economic, cultural, and social impact of regulating all kinds of data flow. Both the COE, in treaty form, and OECD, in guideline form, recognize the importance of uniform data flow regulation among countries. However, a treaty affords a much higher level of protection to data subjects than guidelines. Because European countries already have data protection laws in force, the guidelines stage may have already passed. According to Hondius, "Common guidelines are a useful first step for legal cooperation between states," but since laws are already in force in many countries, "a legally binding treaty is necessary . . . in order to solve conflicts of jurisdiction."<sup>17</sup>

To date, privacy protection legislation differs in scope of coverage, data subjects and enforcement. For example, the U.S. and Canadian Privacy Acts only compel public entities to disclose information held on individuals, while all the other countries with such regulations also compel disclosure by private entities. While Norway, Denmark, Austria and

Luxembourg include legal persons in their privacy legislation, the other countries with privacy legislation do not. While some countries enforce their legislation through private agents at each data bank, others require state licensing or an authorization. This diversity leads to more costly data processing for foreign data companies because they are required to constantly investigate and comply with varied legislation. This will inhibit multinational data transport. In addition, diverse legislation could also require transmission only to those countries with reciprocal protection laws, as in the Luxembourg Act.<sup>18</sup> The only way to alleviate these problems is to establish ground rules for emerging international data networks through guidelines or treaties with the goal of setting up "an international regime of well-defined, preventive rules . . . for the feeding, operation, and use of international data networks to guarantee maximum safety of the systems not only from a physical point of view but also from an information quality point of view."<sup>19</sup>

### C. Arguments on for and against including "legal persons" in privacy protection

#### 1. For inclusion

The primary reason for privacy protection is to assure the individual an opportunity to limit, inspect and correct *personal* information recorded in any data bank. Thus, Swedish legislation contains special controls over information on an individual's political or religious views, whether they have received social welfare or have been treated for alcoholism.<sup>20</sup> The concern is that privacy may be violated by accumulation of information and dissemination where the individual does not want it to go.<sup>21</sup> The proponents of inclusion argue that this individual privacy right can only be preserved if privacy legislation is extended to legal persons.

Luxembourg's new privacy protection legislation states that inclusion is a matter of avoiding discrimination which would result from treating individual and corporate financial information differently:

"It is within the context of this new individual right that must be placed the issue of the citizens protection area. The opinion of the Commission is that a new discrimination must not be created by refusing above listed rights to corporate bodies. This would mean institutionalization of a gap. It is customary for financial establishments to keep information on their customers credit rating. In the event that an inaccurate information would be stored in a banking establishment, the commercial firm which is the bank's customer must, as the case may be, have the right to request correction of data whose inaccuracy has been verified. Could this right to correction be refused to certain categories of persons? The Commission does not think so. This is why it approved the government draft which is widening the scope of the law to natural persons and corporate bodies."<sup>22</sup>

The French first considered inclusion of legal persons because it would provide small firms with the opportunity to find out what big firms knew about them: "Originally, both business and government were favorable to this inclusion

. . . the French employer's association . . . believed it would allow firms to find out what data the government and other state institutions held on them."<sup>23</sup>

Another reason for including legal persons which specifically addresses individual privacy is that in some cases the interests of a natural person can be so intertwined with the interest of a legal person that the natural person's rights can only be fully protected if legal persons are also covered.<sup>24</sup> For example, information concerning a small business' financial situation may reveal the personal financial situation of its president. Also, there is a sense of unfairness in the idea that an individual craftsman is protected but if he incorporates, he is not.<sup>25</sup>

Beyond simply preserving privacy, the data protection legislation tends to control the political and economic side effects of data transport.<sup>26</sup> Nations concerned with preserving national sovereignty, developing their own data processing industry, and controlling cultural influences have and will consider privacy protection legislation, with the inclusion of legal persons, as a means of protecting these interests.

Sovereignty and economic concerns focus on *who controls* data flow. At present, data flow has a lopsided nature. Multinational computer services and the telecommunication industry are primarily located in one place, the United States. Countries interested in developing their own domestic computer services face stiff foreign competition because importing information and services is less expensive than developing local industry. Also, national sovereignty may be threatened when foreign control of data exists because nations fear being cut off from such data.<sup>27</sup> Finally, this technological and economic dependence by smaller countries creates cultural conflicts over the influx of foreign media data, in some cases causing rejection of technology or disruption of social order.<sup>28</sup>

Developing nations are also concerned that the multinationals centered in the U.S. create a serious unemployment problem because data processing takes jobs from the minimally skilled workers in their countries. Any contrary argument that the new industry creates jobs must be careful to distinguish the new, skilled labor jobs created by multinationals providing countries with processed data from the unskilled jobs lost through their replacement by mechanization.<sup>29</sup>

Another economic concern of developing nations is the competition the foreign data industry has on their state-owned Postal, Telephone and Telegraph (PTT) administrations. The issue is that foreign private networks divert their revenues.<sup>30</sup>

All the above economic and political concerns lead nations to consider legislation to prevent these problems. Including legal persons in privacy protection is one way to alleviate some of the problems by regulating the foreign data processing industry to the point of either removing it altogether from their country or severely limiting its influence.

#### 2. Against inclusion

Focusing on the foremost reason for privacy legislation, individual privacy, the proponents of excluding "legal per-

sons" from privacy protection argue that privacy by definition cannot be an attribute of a "legal person" because the nature of individual privacy interests differs from those of business entities. Since privacy legislation concerns itself primarily with the accumulation of personal social data, such as religious and political affiliations, the reasons for protection cannot extend to legal persons, whose data is financial or proprietary. Also, legal persons' "privacy interest" is already fully protected by confidentiality and trade secret legislation.<sup>31</sup>

A special fear associated with privacy protection which includes "legal persons" is that the skillful use of access rights can distort competition.<sup>32</sup> If the right of access is extended to legal persons, it would mean that competitors could find out what was on file and divulge proprietary information.<sup>33</sup>

To the countries which are against inclusion, more control of "legal persons" would *not* encourage individual freedom from intrusion, but would, on the contrary, be the first step in stifling personal privacy by increasing state control of information transmitted in or out of a nation by the private sector,<sup>34</sup> and would ultimately signify government control of commerce and trade,<sup>35</sup> resulting in lower productivity and slower development of the data processing industry.<sup>36</sup>

Another concern is the cost of privacy legislation that includes legal persons. Because European countries have and will, even in the face of the *minimum* COE treaty proposal, enact inconsistent privacy protection laws, multinational TDF companies must spend revenues to determine if their existing and planned record-keeping procedures are lawful.<sup>37</sup>

Also, it is argued that smaller countries have the most to gain from unfettered movement of information across their borders. Since none of the smaller countries is technologically self-sufficient, those that restrict free flow will impede their own economic growth.<sup>38</sup>

Finally, inclusion of legal persons may create "Data Havens" by excluding companies from countries with TDF regulations. Those countries with no laws restricting the use of data would acquire the data banks causing a loss of control of domestic information and revenues in regulated countries.<sup>39</sup>

#### *D. U.S. position on "legal persons": economic and foreign policy compel exclusion*

##### **1. U.S. privacy right and business entities**

Because the right to privacy developed in the United States excludes corporations, the U.S. would be supporting a policy contrary to its own on privacy if it supported inclusion of "legal persons." The legal right to privacy was most clearly articulated in *Roe v. Wade*, 410 U.S. 113 (1972):

"... the Court has recognized that a right to personal privacy, or a guarantee of certain areas or zones of privacy, does exist under the Constitution. . . . These decisions make it clear that only personal rights that can be deemed 'fundamental' or 'implicit in the concept of ordered liberty' . . . (such as) activities

relating to marriage, . . . procreating . . . contraception, . . . family relationships, . . . and child rearing and education." 410 U.S., at 152-153.

In *United States v. Morton Salt Co.*, 338 U.S. 632, 652, 70 S.Ct. 357, 94 L.Ed. 401 (1950), the Supreme Court ruled that corporations have no privacy right equal to individuals and therefore could not resist an FTC subpoena on invasion of privacy grounds:

"[C]orporations can claim no equality with individuals in the enjoyment of a right to privacy. [Citations omitted]. They are endowed with public attributes. They have a collective impact upon society from which they derive the privilege of acting as artificial entities."

The Privacy Act prohibits disclosure of personal records by government agencies on "individuals," U.S. citizens or aliens with permanent residence, except to that individual, and does not provide business entities with the same protection. 5 U.S.C. §552a. Business entities are covered by the FOIA, but not under the "privacy" umbrella. The FOIA requires any government agency to disclose to anyone information held on natural and legal persons except where disclosure would reveal trade secrets or confidential business information. 5 U.S.C. §552(b)(4). This exception to disclosure has been interpreted to cover not only trade secrets, but "any commercial or financial information . . . if its disclosure is likely . . . to cause substantial harm to the competitive position of the person from whom the information was obtained." *National Parks & Conservation Assn. v. Morton*, 498 F.2d 765, 770 (U.S.C.A., D.C. 1974).

Further, trade secrets and commercial or financial information which is privileged or confidential may not be disclosed by any employee of the U.S. in trade negotiations under the Federal Advisory Committee Act, 19 U.S.C. 2155(g)(1)(A), and disclosure of such information by any government employee subjects him to a fine or imprisonment. 18 U.S.C. §1905.

If the U.S. were to recommend inclusion of legal persons in privacy protection, information would be available which is now exempt from disclosure because of the harm disclosure would do to competition. This would contradict the federal position as stated in the FOIA and Federal Advisory Committee Act that business' trade secrets and confidential information must not be disclosed, and would expand privacy protection expressly limited to individuals in the Privacy Act to include legal persons.

The U.S. has limited the scope of privacy protection legislation, by considering the use and nature of the information and how that use and nature corresponds with the U.S. right of privacy. Specific concerns, such as governmental intrusion, which is prevented by the Privacy Act and the FOIA, and financial records, which are available to individuals and small companies which would be most harmed by a secret blacklist, are addressed. But a broad, sweeping control of information in order to protect the right of privacy does not exist. If the U.S. is to be consistent with its domestic privacy protection in its international recommendations, it must use

this sectional approach, examining the use and nature of the information before recommending control over it.

Part of this recommendation, however, must be that legal persons be excluded from privacy regulation to avoid permitting inspection of confidential business information harmful to competition.

## 2. U.S. foreign policy: free enterprise, private participation and cooperation

The U.S. position found in legislation on foreign economic policy further demonstrates that its support of inclusion of legal persons in privacy legislation would be inappropriate.

Congressional policy on international development requires respect for individual civil and economic freedom. 22 U.S.C. §2151(a). Foreign policy goals must include:

“(3) the encouragement of development processes in which individual, civil and economic rights are respected and enhanced; and

(4) the integration of the developing countries into an open and equitable internal economic system.” 22 U.S.C. §2151(a)(3)(4).

Also, free enterprise is explicitly a part of the U.S. foreign policy:

“(a) The Congress of the United States recognizes the vital role of free enterprise in achieving rising levels of production and standards of living essential to economic progress and development. Accordingly, it is declared to be the policy of the United States to encourage the efforts of other countries to increase the flow of international trade, to foster private initiative and competition . . . to discourage monopolistic practices, to improve the technical efficiency of their industry, agriculture, and commerce, and to strengthen free labor unions; and to encourage the contribution of United States enterprise toward economic strength of less developed friendly countries.” 22 U.S.C. §2351(a). (Emphasis added.)

By agreeing to include legal persons in privacy legislation the U.S. would violate its articulated foreign policy goals because it would thereby support the suppression of free flow of trade and private enterprises.

### E. U.S. recommendation: omit legal persons from privacy protection

Clearly, the U.S. interest is in preserving the free flow of data. The U.S. must also encourage a unified TDF policy which considers all the issues, including the economics, sovereignty and independence, of other countries. In view of U.S. domestic privacy legislation, which provides privacy protection only for data categories posing great risks to individual privacy, and its foreign policy, which encourages free flow of data and free enterprise, the U.S. should urge that legal persons be excluded from privacy protection.

## FOOTNOTES

1. Donaghue and Longworth, *Transborder Data Flow*, Paper presented at Computer Security and Privacy Symposium, Scottsdale, Arizona (1978) at 24, col. 2.
2. Internal Revenue Code: IRC §7701(a)(1).
3. 11 U.S.C. §101 (30).
4. 5 U.S.C. §551 (2).
5. 5 U.S.C. §552a(2).
6. 12 U.S.C. §3401 (4).
7. Pantages, *Europe Moves Toward Controlled Data Flow*, *Data-mation*, Nov. 1, 1978, at 82, col. 1; *Second Draft Guidelines Governing the Protection of Privacy in Relation to Transborder Data Flow of Personal Data*, by OECD, Feb., 1979, at 8, permitting guidelines to include:
 

“Groups of persons, associations, corporations or any other bodies whether or not such bodies possess legal personality.” (Omitted in 3rd draft but quoted here to demonstrate a common definition of “legal persons.”)
8. Donaghue and Longworth, *supra* at 24.
9. 5 U.S.C. §551(1), (2), §552.
10. Pantages, *supra* at 80, col. 1.
11. 3.1 Council of Europe Resolution (74) 29 On the Protection of Privacy of Individuals Vis-a-Vis Electronic Data Banks in the Public Sector, which states:
 

“. . . the adoption of common principles in this field can contribute towards a solution of these problems in the member states and can help to prevent the creation of unjustified divergencies between the laws of the member states.” *Id.*, at 1.
12. Pantages, *supra*, at 80, col. 1.
13. *Id.*, at 82, col. 1; Council of Europe Resolution, *supra*:
 

“For the purposes of this resolution, personal information means information relating to individuals (physical persons) . . .” *Id.* at 1.
14. Pantages, *supra*, at 82, col. 1.
15. *Second Draft Guidelines*, OECD, *supra*, at 8.
16. *Third Draft Guidelines*, OECD, April 17, 1979, at 3.
17. Pantages, *supra*, at 81, col. 1.
18. Luxembourg Bill Regulating Personal Data Use in Data Processing, March, 1979 Chapter 2, sec. 1, art. 5(2); P. Hirsh, *Europe's Privacy Laws—Fear of Inconsistency*, *Data-mation*, Feb., 1979, at 85, col. 3.
19. Hirsh, *supra*, at 85, col. 3, quoting Hans Peter Gassman, on the architects of the OECD draft guidelines.
20. Swedish Data Bank Statute (1978:289) of May 11, 1973, §4.
21. Canadian Department of Communications and Department of Justice, *Privacy & Computers, Task Force Report*, Ottawa, 1972 at 1, 14.
22. Luxembourg Bill, *supra*, Report at 8.
23. Transnational Data Report on New French Laws on Information Processing and Freedom, at 3.

- 
24. A.F.I.P.S. Panel on Transborder Data Flow, April 10, 1979 Notes of Meeting at 1.3(A).
  25. R. Turn, *Notes on Legal Persons*, April, 1979, at 5.
  26. E. J. Novotny, *Economic Aspects of Transnational Data Flows*, Insert to Chapter 5, AFIPS TBDF Panel Report, May, 1979 at 1; France's Minister of Justice, Louis Joinet, emphasizes the importance of trade aspects of TDF:

"Information is power and economic information is economic power. Information has an economic value, and the ability to store and process certain types of data may well give one country political and technological advantage over other countries. This, in turn, may lead to a loss of national sovereignty through supranational data flows." Donaghue and Longsworth, *supra*, at 23.
  27. A.F.I.P.S. Panel on Transborder Data Flows, *Overview of TDF Issues*, W.G.3, May 1979, at 3-7; Novotny, *supra*, at 5.
  28. A.F.I.P.S. Panel, W.G.3, *supra*, at 3; J. Eger, *Transborder Data Flow*, *Datamation*, Nov. 15, 1978, at 50, col. 3.
  29. F. Lamond, *Europeans Blame Computers*, *Datamation*, Nov. 1, 1978, at 107, col. 3; Novotny, *supra*, at 7.
  30. Novotny, *supra*, at 5.
  31. R. Turn, *supra*, quoting Joinet at 5.
  32. *Id.*
  33. *Transnational Data Report on New French Law*, *supra*, at 3.
  34. Hirsh, *supra*, at 85, col. 3; Novotny, *supra*, at 2.
  35. Donaghue and Longsworth, *supra*, at 24.
  36. *Datamation*, *Canada's Economic Concerns*, Nov. 1, 1978, at 67, col. 3. Duties on foreign computer equipment bought by Canadian companies make domestic development of data processing expertise too expensive, so the Canadian computer industry is against government protection intended to limit U.S. competition.
  37. Hirsh, *supra*, at 85, col. 3.
  38. *Id.*, at 87, col. 2.
  39. Eger, *Transborder Data Flow*, *Datamation*, Nov. 1, 1978, at 52, col. 2.



## Simulation

Simulation is one of the oldest application areas for computers. Traditional users of simulation include the aerospace, military/defense, process control, industrial control, and energy industries. Newer users of simulation include corporate management, private and governmental planning agencies, and small businessmen. The NCC sessions describe advances in traditional areas such as process control as well as new applications such as the modeling of computer software, decision support systems for business management, futures planning, and small business decisionmaking. A particularly significant growth area is the use of simulation to explore the effects of decisions and alternative futures. This area is represented by sessions on decision support systems, cross-impact models, and small business applications.

The simulation of complex engineering systems has long been a major application of computers. Tremendous activity has occurred in flight simulation, the analysis of aerospace and military systems, the modeling of chemical and physical processes, and the design of power plants, reactors, and other large, complex systems. This activity is continuing and is being affected by new computing techniques, new computer languages, and new software packages. The session on "Advances in Process Control" investigates the effects of new hardware, software, and methodology on the process control industry. The paper by Gordon and Robinson on "Using preliminary Ada in a process control application" describes the use of the U.S. Department of Defense standard language ADA in process control. The paper by Sagues on "Computer aided heat penetration tests for the food canning industry" describes the use of small minicomputers in test automation.

Meanwhile, computers themselves have created new modeling problems. The modeling of computer performance is a well established application area. The modeling of computer software, a newer area of interest, is being explored in a panel session entitled "Software Models: History, Current Status, and Future Directions." Topics of concern will include reliability and quality assurance models, model validation, and the needs of the software community.

Among the more intriguing applications of simulation is its use in predicting the future course of events or at least exploring the effects of alternative decisions or alternative time histories. This use becomes particularly important in a time of rapid change and great uncertainty. Decision support systems provide top management with the ability to explore the effects of decisions with a reasonable expenditure of time and money. User-oriented planning languages allow rapid formulation of problems in a comprehensible form. The panel session on "Decision Support Systems" describes the effective use of decision support systems, the advantages of modern planning languages, and the importance of decision support systems in data processing organizations.

The session entitled "Simulation—A Planning Tool" describes the use of cross-impact models in long-range planning. Cross-impact models are based on the effects of events on each other—that is, whether the occurrence of a particular event increases or decreases the probability of the occurrence of other events and by how much. The most fully developed cross-impact model is the INTERAX world model developed by the University of Southern California's Center for Futures Research. This session will include an overview of cross-impact models and a description of the INTERAX model, emphasizing its goals, methods, and uses. The paper by Rosenthal entitled "A cross-impact simulation forecast of the data processing industry" describes the use of the INTERAX model in simulating the evolution of the emerging data processing industry.

New and less expensive computers allow the use of simulation in areas where it was formerly too expensive. Small business is a primary example, since minicomputers and microcomputers now provide low-cost computing power in an area with the same basic problems as larger businesses. Typical applications of simulation will be explored in a session entitled "Simulation in Small Business." Topics to be explored include strategic systems modeling, investment analysis, business forecasting, and the presentation and analysis of simulation results.

Lance A. Leventhal  
*Area Director*





# Using preliminary Ada in a process control application

by M. E. GORDON and W. B. ROBINSON

*The Foxboro Company*  
Foxboro, Massachusetts

## INTRODUCTION

This section contains background information on the Ada language definition process, an introduction to features of Ada, and an overview of the Model Controller Operating System (MCOS), which was coded in Ada. More detailed information on Ada can be found in *The Ada Language Reference Manual*,<sup>1</sup> *The Ada Language Rationale*,<sup>2</sup> and *Programming with Ada*.<sup>3</sup>

### *An Ada introduction*

#### **Background**

Ada is the programming language being developed under the auspices of the Department of Defense. The language development has extended over a period of several years, from requirements specification (Strawman, Woodenman, Tinman, Ironman, Steelman) to language definition. Accompanying the language development is the specification of a language support environment (Sandman, Pebbleman, Stoneman), which is progressing closely after Ada itself.

Ada is being considered as a standard language for process control by the Long Term Programming Languages (LTPL) Committee of the Purdue Workshop and the European LTPL Committee. Both ANSI and ISO standardization efforts are being initiated.

Currently in a formative stage, Ada is undergoing revisions in response to a Test and Evaluation (T&E) of the language. The programming project described in this paper was undertaken as part of the T&E review, and is based on the preliminary Ada language definition of June 1979. Since there are, as yet, no Ada compilers available, the evaluation is static. However, the Ada features described below are intrinsic to the conceptual model of the language and are unlikely to change.

This section will discuss how Ada language features support the modern language concepts of: (1) data abstraction, (2) modularity, (3) encapsulation, (4) concurrent programming, (5) machine independence, and (6) orthogonality and extensibility.

In later sections, it will be shown how such language features may be applied to a typical programming problem in a process control systems application.

#### **Data abstraction**

Data abstraction is supported in Ada by programmer-definable data types. Type declarations collect knowledge of common properties of objects in one place, thereby facilitating software maintenance. The principal advantage is greater software reliability, because the programmer's code is closer to the expression and solution of the applications problem.

Ada's strong typing is based on name equivalence, rather than structural equivalence. No implicit type conversions are allowed. Explicit conversions can be used in some cases, for instance, in converting numeric types. In other cases, Ada provides the `UNSAFE_CONVERSION` function as an escape hatch.

For programmers accustomed to creating variables on-the-fly (as in FORTRAN), Ada's requirements for declaration of variables and types may seem overly restrictive at first. With proper use of Ada's data facilities, however, the benefits far outweigh the constraints.

#### **Modularity and program structure**

Modularity is supported not only by traditional subprograms (procedures and functions), but also by Ada modules (packages and tasks). Although the overall structure of an Ada program follows the conventional block structure of ALGOL 60, it differs in that modules may be separately compiled and arbitrarily included at various levels of the program hierarchy. Ada offers control over visibility and scope through *restricted* clauses, which may override inheritance rules of module nesting. Importing is done by program unit name, not by object name.

#### **Encapsulation**

Package modules are the cornerstone of the language. Through them, encapsulation of data with their associated operations is possible. Other uses of packages are grouping related procedures together and forming a collection of related types and data objects.

Packages consist of a specification part and an optional package body. The specification part of a package contains

the logical interface (e.g., type declarations, procedure specifications) which other modules and subprograms may access. The package body contains the implementation of the operations specified in the visible part. The package body is a mechanism for information-hiding, that is, concealing implementation details from the users of the package.

Ada's separate (but not independent) compilation reinforces the realization of encapsulation via modularization. Compilation checks between compilation units are the same as those within a unit. Therefore, it is not necessary to wait until link time to discover most interface errors. This checking will be accomplished through a library management system, which is a requirement of the Ada support environment.

### Concurrent programming

Tasks, similar to packages in format, are the means for implementing concurrent processes. Unique to the Ada tasking mechanism is the rendezvous concept, which serves the dual functions of synchronization and communication in a parallel processing environment. Three controversial features related to Ada's tasking are scheduling, interrupt handling, and access to shared data. They are likely to change in the final language definition.

### Machine independence

Machine independence is the ultimate goal of a high-level language such as Ada. Ada language features, as described above, allow the systems programmer and, later, the maintenance programmer to operate at a level which is closer to the system specification than is possible with older languages like FORTRAN (developed in 1954). Actual machine-dependent code may be kept to a minimum. The result is a more reliable, robust software system that is easier to develop and maintain, consistent with the conventional wisdom of software engineering.

### Orthogonality and extensibility

Ada offers language primitives and rules for combining them (orthogonality<sup>4</sup>) to build specialized structures and functions. This principle of orthogonality and the related principle of extensibility are demonstrated by the I/O packages, which are written in Ada. Although lacking some key features (e.g., variable-length strings, bit types), Ada provides the tools to create them. This is a critical requirement for a systems implementation language.

Such freedom in a language exacts a price, that price being the responsibility for a programming discipline. When properly used, Ada can be a powerful tool for both systems and applications software development. However, without the appropriate programming methodology, the benefits of Ada are easily lost. For example, arbitrary use of the exit statement to create unstructured loop constructs would be a misuse of the language.

In the following sections, it will be shown how Ada may be applied properly to an embedded operating system for process control. In particular, the remaining discussion will illustrate how Ada influences all stages of the software development process, from initial design to final implementation.

### The model controller operating system

The model controller is a system that allows digital implementation of multiloop control systems based on traditional (analog) control diagrams. It was conceived as a typical example of a small, real-time system providing these main features: (1) communications interface to host computers, (2) control algorithms and control database based on block diagrams, and (3) interface to process input/output components. Each of these is discussed in detail later.

The Model Controller Operating System (MCOS) supports these features with the multi-tasking facilities provided by Ada. Figure 1 shows a simplified sketch of the model controller.

One approach to reliability in digital control systems is redundancy. In the model controller, this is reflected in the dual-controller, dual-port, dual-link architecture. In the event of a failure of any main system component, the backup component is automatically activated. The process database is regularly (every control cycle) transferred to the backup, or tracking, controller via the shared data buffer. With appropriate checks against contaminating its database, the tracking controller is prepared to take control with a very recent database copy. The two controllers also share access to the process I/O bus. The hardware redundancy is complemented by a software redundancy: both controllers have their own copy of the MCOS software, which will be described in the following sections.

The communications link allows host computers to configure or to change the configuration of the control database, to read or write particular values in the database, and to monitor or change the status of each of the controllers.

The process I/O bus, which can be accessed by only one controller at a time, allows the controllers to read or write analog and digital input/output devices.

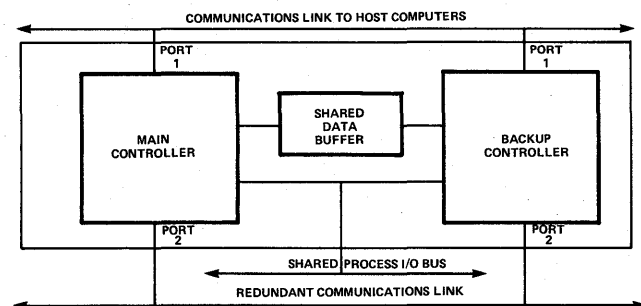


Figure 1—Model controller.

### Overview of the operation sequence

At every quarter second clock interrupt, each controller begins by checking its mode. Typically one is on control, the other tracking. The lead controller processes all control blocks in the following manner: All process I/O components are read, conditioned, and stored in a table.

For each block: (1) appropriate parameters (such as measurement or set point) are updated; (2) the algorithm for each block is executed; (3) if needed, an output is sent from the block to the appropriate component.

After all blocks are processed, the database is written into the buffer.

The tracking controller copies the data buffer into its own database.

At any time, the system might receive an interrupt on the communications channel. Typical communications messages supported are: (1) secure/release, (2) read/write database, and (3) read/write controller status.

When neither the control processing nor the communications task is active, a security task executes, monitoring the health of the system.

### SYSTEM DESIGN WITH ADA

This section contains discussions of some of the design decisions made and relates them to the facilities of Ada for modularization and for representation of data. The design issues are: (1) modularity and program structure, (2) data structures and representations, (3) exception handling, (4) scheduling, and (5) interrupt handling.

#### Modularity and program structure

The overall program structure of the Model Controller Operating System (MCOS) is illustrated in Figure 2a. The main procedure contains the module specifications of the three primary tasks: communications, executive, and security. The function of the main procedure is to initiate the three tasks. The code for the main procedure is straight forward, as shown in Figure 2b.

Each task body is a separate compilation unit. By separating the task body from its specification, implementation details may be changed without necessitating a recompilation of the main procedure (assuming the interface remains the same). Ada's separate compilation facility was used extensively in the design of MCOS, to take advantage of the logical interface—physical implementation separation. This

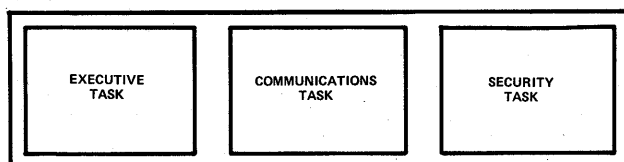


Figure 2a—Main procedure contains specifications of the three primary tasks.

#### PROCEDURE MAIN IS

TASK COMMUNICATIONS;

TASK EXECUTIVE;

TASK SECURITY;

TASK BODY COMMUNICATIONS IS SEPARATE;

TASK BODY EXECUTIVE IS SEPARATE;

TASK BODY SECURITY IS SEPARATE;

BEGIN

INITIATE COMMUNICATIONS, EXECUTIVE, SECURITY;

END MAIN;

Figure 2b—Main procedure of MCOS.

facility supports top-down design in that the logical interfaces may be defined first, with the implementation stubs developed later. Another advantage of the separation principle is that it streamlines the code, improving the readability of Ada program units.

Within each task body, the specifications for its internal packages are local, while their bodies are separate. The tasks and their primary functions are: (1) communications task: handle port interrupts; (2) executive task: process control blocks (if main controller) or track the database (if backup controller); and (3) security task: perform various software checks during any idle time.

The communications and executive tasks are discussed more fully in later sections.

The final two main modules of MCOS are packages. The database package and status.manager package are separate compilation units in the MCOS program library, and are independent of the overall block structure. The purpose of the status.manager package is to encapsulate global type and data declarations with their associated operations. The database package, on the other hand, contains related types and objects for global use and a synchronizing semaphore task. The two packages are self-sufficient in that they see no other units in the program library. Other program units which require access to the global data make it visible by including the package name in a visibility list. Therefore, importing of global data is done on a module basis, rather than on a single variable basis. The global packages may be imported at any level of the block structure hierarchy.

As demonstrated in Figure 3c, good programming practice dictates explicit import lists of objects to be listed in comments. Without such information, the code is virtually unreadable, since it is not apparent which data objects are being referenced. Rather than burden the programmer (who may be inconsistent and/or error-prone), such information could be generated automatically by the compiler or text editor.

It must be noted that the MCOS program structure presented here was achieved through several iterations. Because of the novel interplay between traditional block structure and separately compiled modules, classic rules-of-thumb for structured design were not directly applicable. Ada is a language without a history and, consequently, with-

out a refined programming methodology. Proper use of Ada features was discovered partly through trial-and-error. A good program structure for MCOS was achieved by following certain guidelines: (1) control visibility as much as possible by using traditional block structure; (2) within the general block structure of the program, textually include only the module specifications; use stubs for module bodies, which should be developed and compiled separately; (3) reserve use of separately compiled modules for global library packages; (4) whenever feasible, compile the specification part and the implementation part separately to reduce re-compilation dependencies.

### *Data structures and representations*

To quote Wirth,<sup>5</sup> "The choice of (abstract) representation of data is often a fairly difficult one, and it is not uniquely determined by the facilities available. It must always be taken in light of the operations to be performed on the data." In our model system, there are several instances where the choice of abstract representation was complicated by the fact that two different tasks required access to the same data objects. In another instance, to be described later, the strong type restraints of Ada had to be circumvented to allow a more flexible approach to handling raw data for input and output.

The largest data object, the array of control blocks, is considered in detail below. The design of data structures proceeded in parallel with the overall module design.

### **Shared data**

The critical issue here is the need to provide adequate protection of the database, which can be accessed by both the communications task and the control executive task.

In MCOS, total encapsulation of the database was rejected for two reasons: first and foremost, the number of transactions using the database is prohibitively large, and using specialized interfaces would degrade performance. In particular, the communications task has a 10 millisecond time-out period and thus cannot be delayed too long by control processing. Secondly, a small dedicated system does not require the level of protection encapsulation offers. The communications task performs sufficient validity checks to protect the database in transactions with host computers.

Although encapsulation of the database was rejected, mutually exclusive access to the database must be provided so that consistent data is used by the tasks sharing access. The simplest solution, and the one finally used, employs Ada's generic semaphore task to create a critical region, during which only one task can access the database. It may be that this solution is too inefficient, in which case an equivalent solution would have to be implemented in machine code.

By way of contrast, the package `status_manager`, described below, is an example of the use of the package structure for encapsulating data objects and the operations per-

formed on them. The brief duration of the exclusive read/write access to the status registers makes encapsulation feasible.

### **Physical representations of data objects**

Since the object machine for the Model Controller Operating System is predicated as having a small memory, space issues are important. Often a record has several components. To allow the compiler to assign the storage for these may use more storage than is desirable. Ada provides several representation facilities for records<sup>1</sup> (Chapter 13): One can specify the number of bits to be used representing objects of a given type, specify that the compiler is to use packing, or specify actual word and bit layouts for components of a record. All of these facilities were used in the MCOS exercise.

### *Exception handling*

In any real-time system, security and robustness are essential. Exceptional conditions, such as overflow during a calculation, should not cause a system crash, but must be handled in a meaningful way so the system can recover and continue processing. Ada provides an exception handling facility which appears to be adequate, although sometimes cumbersome<sup>1</sup>(11). Examples of the use of exception handlers are given in later sections.

### *Scheduling*

The strategy for scheduling the tasks and procedures in a real-time system must be carefully thought out and carefully implemented. In the MCOS, the communications task must execute immediately upon receipt of an interrupt. Other scheduling requirements are somewhat "softer." However, it is in the area of scheduling that Ada seems to have the greatest weakness. According to the LRM<sup>1</sup> (9.8), "The language does not specify when a scheduling decision is made; for example, a round-robin time-sliced strategy is acceptable." There is a language-defined priority attribute for tasks which can be used in scheduling decisions. However, there seems to be little in the language to facilitate the design and implementation of a scheduling algorithm.

MCOS requires a scheduler with the following properties: (1) the scheduler is to be invoked when a new task enters the ready queue, in particular when an entry call or interrupt occurs; (2) when the scheduler is invoked, ready queues are examined and tasks with the higher priorities are executed first.

### *Interrupt Handling*

Interrupt handling is another area where Ada, as currently defined, falls far short of the mark. This is a critical area for real-time applications, such as process control, in which cer-

tain hardware interrupts demand immediate, and uninterrupted, service. In MCOS, for example, a communications interrupt requires that the message be processed and a reply sent within 10 msec, otherwise a timeout occurs, putting the controller on standby. Unfortunately, there is no way to guarantee dedicated resources to a high-priority interrupt in Ada.

Ada does not distinguish between hardware interrupts and software signaling between tasks. An interrupt is mapped onto a rendezvous entry, via a representation specification. Yet, there is no way to indicate the urgency of an interrupt on the entry queue. Task priorities only determine which of several tasks waiting on the ready queue will be serviced next. However, entry queues are handled strictly on a first-in, first-out basis.

The Ada language design team has recognized the inadequacy of the current interrupt handling mechanism. It is expected that the problem will be rectified in the final language definition (June 1980). Therefore, in the MCOS programming exercise, the communications interrupt was coded in Ada for illustration purposes, with no assertion of correctness.

## ADA IMPLEMENTATION OF MCOS

This section provides a more detailed view of some of the issues that were raised earlier. In particular, using Ada in the actual implementation of MCOS is discussed with regard to the database, control processing, and communications processing.

### The database

#### Control blocks

The primary function of the model controller is performed through the control blocks and their corresponding algorithms. In a typical process control application, several types of control algorithms, such as proportional-integral-derivative (pid), lead-lag (llag), etc., would be used. MCOS has the pid, llag, nonlinear (nonl), and digital input (din) algorithms as a suitable cross-section. Each system, however, can have up to 32 control blocks, of which an arbitrary number can be pid, an arbitrary number can be llag, and so on.

In a particular control scheme, blocks can be interconnected, can obtain inputs from process I/O devices, and can generate outputs for process I/O devices. A sample configuration is shown in Figure 3a. Because of this interconnectability, blocks are processed in sequence. In general, a block will obtain inputs only from blocks that have been processed before it. This ordering stems from the traditional digital implementation of continuous analog control.

A standard assembly language or FORTRAN implementation of such a system would have to treat the 32 blocks as a massive array of words of undifferentiated type, and the layout of parameters within different block types would be contained in an external document, presumably a system

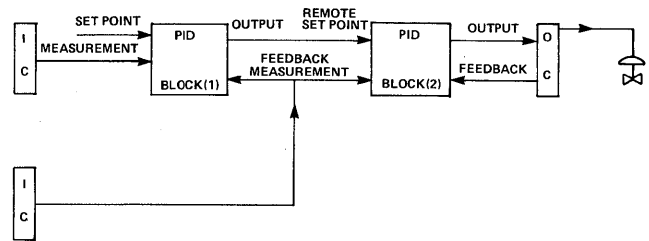


Figure 3a—A sample control scheme.

specification. Hence, the meaning of a particular word in the database would be obscured and opportunities for errors by both original writers and later maintainers is increased.

With the tools of Ada or any other sufficiently typed language, such a situation can be avoided, and the form and meaning of the database items can be given explicitly in the program itself. Variant records were chosen to represent the control blocks. (A variant record is a record with choice of alternative substructures based on the value of a discriminant component<sup>1</sup> (3.7)). Two rules which affect the utility of variant records are: the discriminant can be changed only during a complete record assignment; and the same component name cannot be used in different variant parts. Both rules caused some difficulties, as will be discussed later.

An enumeration list of block types is the discriminant component for the variant record type. This has the declaration:

```
type block_name is (null_block, pid, nonl, din, llag);
```

The other components common to all blocks were the block status word, the name fields, the options word, and the block parameter. These required separate type declarations and representation specifications as well. In specifying the variants, it was necessary to identify common types that apply to the components of the different blocks. The four major types were value, logical, value-pointer, and logical-pointer. Each is a record in its own right, with a representation specifying one word of storage. For instance,

```
type value is
  record
    from_pointer: BOOLEAN;
    value_is_bad: BOOLEAN;
    counts: normalized_counts;
  end record;
for value use
  record
    from_pointer: at 0*WORD range 0..0;
    value_is_bad: at 0*WORD range 2..2;
    counts: at 0*WORD range 3..15;
  end record;
```

Examples of components in a pid block which are of type value are the measurement, set point, and output. On the other hand, in the din block, the measurement and output are of bit\_pattern type, and there is no set point. Hence it was necessary to place these components in the variant part of the control block.

The final problem was to choose a naming scheme that allowed easy use of the control blocks in the algorithms. Separately named components would have led to tedious implementation of the algorithms. Instead, like parameters were grouped into arrays by type, with each array indexed by an appropriate enumeration type. For instance,

```
type pidlist is (meas, remote.sp, feedback, halim, lalim,
  hdlim, ldlim, holim, lolim, bias, pband, rate, integral,
  setpoint, output, absdb, devdb, outdb, kl,
  filtered_meas, integral_balance);
type llaglist is (meas, dynamic_gain, timel, bias, output);
```

Note that these lists overload literals, such as *meas*, and hence, when ambiguities arise, care must be taken in using them, for instance by writing "pid\_list(meas)" explicitly.

Finally the data object *block* is declared in the database package by

```
block: array (1..32) of control_block;
```

While the development of these types proceeded in top-down fashion, Ada has the unfortunate and annoying restriction that the type declarations must be presented in bottom-up order. This is a hindrance to both writers and readers of a program.

#### Status manager package

MCOS contains status registers to indicate certain conditions of the system hardware/software. The logical representation of the status flags and the specifications of the available operations are encapsulated in the visible part of the *status\_manager* package. The physical representation of the registers (Boolean arrays), as well as the implementation of their corresponding access routines, are concealed in the package body. The *status\_manager* package is global to MCOS. Its visible part provides a simple interface for accessing the status registers.

Within the package body, the implementation of the various access routines differ in the level of protection afforded to the status registers. Protection mechanisms are provided only as dictated by the functional requirements. For example, since writing to the controller status register is accomplished via hardware command registers, no extra protection is needed. On the other hand, since the unit status register is directly read/write accessible, a high degree of protection is desirable. A server task provides this protection. Here the rendezvous is used to prevent simultaneous access to the unit status register by parallel tasks:

```
package body status_manager is
  task body protect_status_reg is
  begin
    loop
      accept set_status (flag: unit_stat_list_change;
        new_stat: Boolean) do
        unit_stat_reg (flag) := new_stat;
```

```
      end set_status;
    end loop;
  end protect_status_reg;
begin
  initiate protect_status_reg;
end status_manager;
```

Note that the *initiate* statement of the server task is placed in a *begin* block at the bottom of the package body. Task initiation occurs when the package body is elaborated at run-time.

Since Ada does not prevent simultaneous access to shared data by parallel tasks, it is the programmer's responsibility to ensure the proper level of protection by controlling access via the rendezvous or semaphore. Ada provides protective mechanisms, but does not enforce their use. For system reliability in a parallel processing environment, therefore, good programming discipline is required. Without it, system security is threatened.

#### Control processing

##### The controller executive

The controller executive is a task that executes in parallel with the communications task. The executive accepts a clock interrupt to begin the control cycle and determines the controller's current mode (e.g., standby, control, etc.). A controller that is tracking reads the database buffer. A controller in standby simply exits. When the controller is in control mode or initializing mode, it runs a sample control algorithm and compares the output to a known result. If this checks, it proceeds to control block processing. Otherwise it takes itself off control and exits. The executive code is given in Figure 3b.

##### The control package

The procedure *do\_control\_processing* is in the module *control\_package* and is called from the task executive. It handles block initialization and regular control processing in a uniform manner. Since blocks can be taken off control by a host computer, this must be checked by reading the appropriate block status bit. The control package is hierarchical in organization since no parallel processing occurs within it.

Two difficulties encountered in control processing involved type conversions and exception handling. We discuss these in detail below.

##### Process I/O components

In the MCOS there are 100 input/output components, each of which can be one of several types. The first attempt at representing the components used an array of variant records. However, physical limitations required using the iden-

```

RESTRICTED (MAIN, STATUS_MANAGER)
SEPARATE TASK BODY EXECUTIVE IS
  USE MAIN, STATUS_MANAGER;
  -- EXECUTIVE HAS THE FOLLOWING IMPORT LIST OF OBJECTS.
  -- READ_BUFFER AND WRITE_BUFFER ARE IMPORTED FROM DATA_BUFFER_MANAGER
  -- SET_STATUS AND THE LITERALS OK AND COMP_CHK_BAD ARE IMPORTED
  -- FROM THE STATUS_MANAGER.

  TYPE PROCESSOR_MODE IS (TRACKING_MODE, STANDBY_MODE, INIT_MODE,
                          CONTROL_MODE);
  SUBTYPE ZERO_ONE_OR_TWO IS INTEGER RANGE 0..2;

  ENTRY CLOCK_INTERRUPT;

  INIT_COUNT: ZERO_ONE_OR_TWO;
  CONTROLLER_MODE: PROCESSOR_MODE := STANDBY_MODE;

  TASK DATA_BUFFER_MANAGER IS
  -- THE TASK WHICH COORDINATES ACCESS TO THE DATA BUFFER,
  -- WHICH IS USED TO TRANSFER THE DATA BASE FROM THE "ON"
  -- CONTROLLER TO THE TRACKING CONTROLLER.
  ENTRY READ_GRANT, WRITE_GRANT; --HARDWARE INTERRUPTS
  ENTRY READ_BUFFER; --CALLED BY THE TRACKING CONTROLLER
  ENTRY WRITE_BUFFER; -- CALLED BY THE "ON" CONTROLLER
  END DATA_BUFFER_MANAGER;

  PACKAGE CONTROL_PACKAGE IS
  FUNCTION SAMPLE_PID_TEST_OK RETURN BOOLEAN;
  PROCEDURE DO_CONTROL_PROCESSING (INITIALIZATIONS_REQUIRED:
                                   IN OUT ZERO_ONE_OR_TWO);
  PROCEDURE MODE_CHECK (MODE: IN OUT PROCESSOR_MODE;
                       INIT_COUNT: IN ZERO_ONE_OR_TWO);

  -- THESE ARE THE THREE VISIBLE ENTRY POINTS INTO THE CONTROL
  -- PROCESSING PACKAGE. THESE SUBROUTINES ARE CALLED FROM THE TASK
  -- EXECUTIVE AND FROM NO OTHER.
  END;

  FOR HALF_SECOND_CLOCK USE AT 16:ffc;

  PACKAGE BODY CONTROL_PACKAGE IS SEPARATE;
  TASK BODY DATA_BUFFER_MANAGER IS SEPARATE;

  BEGIN
  INIT_COUNT:=2; --ALL BLOCKS ARE INITIALIZED TWICE
  --AT STARTUP

  LOOP
  ACCEPT CLOCK_INTERRUPT;
  SET_WATCHDOG_TIMER; --THIS REQUIRES ASSEMBLER LANGUAGE CODE
  MODE_CHECK (CONTROLLER_MODE, INIT_COUNT);
  CASE CONTROLLER_MODE OF
    WHEN TRACKING_MODE =>
      READ_BUFFER;
    WHEN STANDBY_MODE =>
      NULL;
    WHEN INIT_MODE:CONTROL_MODE =>
      IF SAMPLE_PID_TEST_OK THEN
        DO_CONTROL_PROCESSING (INIT_COUNT);
        IF INIT_COUNT = 0 THEN
          WRITE_BUFFER;
        END IF;
      ELSE SET_STATUS (OK, FALSE);
        SET_STATUS (COMP_CHK_BAD, TRUE);
      END IF;
    END CASE;
  END LOOP;

  EXCEPTION -- A GENERAL EXCEPTION HANDLER TO SET THE CONTROLLER
  -- TO STANDBY IF AN ERROR IS PROPAGATED FROM A
  -- LOWER LEVEL.
  WHEN OTHERS =>
    SET_STATUS (OK, FALSE);
    CONTROLLER_MODE:=STANDBY_MODE;
  END EXECUTIVE;

```

Figure 3b—Executive task body.

tification field (id) as a flag to indicate absence of an input. This is incompatible with the use of the id as a discriminant. Hence, the use of variant records was rejected.

The solution adopted was to treat each I/O component as a (non-variant) record with two fields:

```

type pio_data is
  record
    id: pio_type;
    twelve_bits: boolean_array (1..12);
  end record;

```

Now the id field can be revised independently of the twelve bit value. Moreover, in the case of the digital com-

ponents, slice assignments can be used to get the information into the appropriate block components. But what happens when the twelve bits must be treated as an integer? Here the package UNSAFE\_PROGRAMMING comes into play. It provides a generic facility for converting between otherwise incompatible types; for instance, for converting an object of type pio\_data to an INTEGER occupying a 16-bit word.

The following representation ensures that each pio\_data value occupies one word.

```

for type pio_data use
  record
    id: at 0*WORD range 0..3;
    twelve_bits: at 0*WORD range 4..15;
  end record;

```

Now UNSAFE\_CONVERSION can be used to translate the single word of pio\_data type to a single word of INTEGER type. This requires the instantiation's *function* data\_to\_int is new UNSAFE\_CONVERSION (pio\_data; INTEGER); temp\_data: pio\_data; followed by the conversion statements temp\_data := (no\_data, pio\_data.twelve\_bits); raw\_count := data\_to\_int (temp\_data);

### Numeric computation and error handling

During the processing of the control blocks, some of the algorithms require numeric computations. Because of the real-time nature of the controller, any error conditions that could arise must be handled in such a way that the system does not halt. This section contains a discussion of the error handlers used in designing the MCOS algorithm set.

Suppose we have three variables given by the declaration

```
X, Y, Z: INTEGER range 0..4000;
```

followed by three assignment statements, where it is assumed that the expressions on the right-hand side yield INTEGER values:

```

begin;
  X: = expression_1;
  Y: = expression_2;
f21 Z: = expression_3;
end;

```

If one of the expressions has a valid INTEGER value outside the range 0..4000, the RANGE\_ERROR condition is raised. At this point the program checks whether a handler has been included within the block. If so, the action specified in the handler is taken. If not, the search for a handler continues in the next outer scope. In the scope given even if we include an exception handler, the program will resume execution not at the next statement following the one raising the exception, but at the statement following the end of the block. Thus, in general, it is not possible to resume execution from the point of error.

In some problems, such as signal conditioning, one wishes



to clamp the value that is out of range. In that case, the following seems cleanest for eliminating the RANGE\_ERROR exception.

```
X:= MAX (MIN (expression_1,4000), 0);
Y:= MAX (MIN (expression_2,4000), 0);
Z:= MAX (MIN (expression_3,4000), 0);
```

However, this will be valid only so long as the expressions on the right are valid INTEGER values. If one of them is not, an OVERFLOW or a DIVIDE\_ERROR exception occurs in the expression evaluation. DIVIDE\_ERROR can be avoided by testing the denominator beforehand. This leaves but two possibilities for OVERFLOW: (1) write the expression as a function within which error handlers are implemented; or (2) enclose each statement in a block with an exception handler, as in this block:

```
begin
  X:= expression_1;
  exception
  --no matter what goes wrong, clamp
  when OVERFLOW =>
    X:= 4000;
end;
—etc—
```

Both of the techniques were useful. In some circumstances it was possible to determine a priori that only RANGE\_ERROR could occur and then the explicit clamping was used. In other cases, the special scope was inserted to localize the error handling.

### Communications processing

The model controller may receive communications interrupts at any time from either port. In MCOS, the communications task functions at the highest priority level to service such interrupts. As discussed in a previous section, the Ada mechanism for interrupt handling is inadequate, and is in the process of being revised by the language design team. By handling the communications interrupts in preliminary Ada, there is no way to guarantee that they will receive the immediate and dedicated attention that is demanded.

In addition to handling port interrupts, the communications task consists of the following units: (1) message\_buffer\_manager package encapsulates the input and output buffers together with the access routine, message\_handler; (2) message\_handler routine decodes the incoming message and calls the appropriate subroutine to process the message and send the reply; (3) error\_counter\_manager package contains the hardware registers (which record the occurrence of transmission errors), and corresponding software access routines.

### Hardware dependencies

The communications process is, perhaps, the most difficult to design and program because of the many direct con-

nections between software and hardware. There are certain circumstances which require a machine code insertion in the Ada program to provide a high-level interface between the hardware and the rest of the software implementation. One advantage to Ada is that it permits such machine-dependent code to interface with the high-level code, isolating and minimizing the degree of machine dependencies. For example, a machine code routine is required to reset the watchdog timer to avoid a timeout.

### Decoding messages

The communications messages which are implemented in MCOS are grouped into three categories: station messages, task messages, and process I/O messages. MCOS responds to messages received from the host computer, but does not initiate them. Station messages involve retrieving status information about the controller, and getting/resetting transmission error counters. Task messages allow the host to switch the controller into tracking or standby modes. Process I/O messages get/set values in the database of control blocks.

Each message has a specific command code which indicates the content of the message. The command codes are implemented by a representation specification for elements of an enumeration type, where elements of the type are assigned internal codes corresponding to values of command codes, as shown in Figures 3c and 3d. Ada supports the principle of separation of logical properties from physical properties. However, in the case of an enumeration type, the ordering of elements in the logical specification must correspond to the ascending numerical values assigned in the representation specification. Yet, despite this dependency, Ada enforces a textual separation in that all associated representation specifications must follow the logical specifications in the declarative part.

The incoming message buffer is an array of bytes. The first part of the message containing the command code must be decoded before the rest of the message can be processed. The decoding was implemented in MCOS via UNSAFE\_CONVERSION of the appropriate bytes into the command code enumeration type. When a case is done on the command code, illegal codes are caught by the *when others* alternative.

### Processing messages

Legal commands are processed by their respective subroutines, whose stubs are internal to the message\_handler procedure. The procedures themselves are separately com-

#### TYPE COMMAND IS

```
(GET_STATUS, GET_ERROR_CTRS, READ_DATA_STANDARD,
 RESET_ERROR_CTRS, STANDBY, STARTUP, SELECT,
 SECURE_RELEASE, SET_RESET_HOLD);
```

Figure 3c—Logical specification of command enumeration type (ordering of elements is determined by ordering in Figure 3b).

FOR COMMAND USE

```
(GET_STATUS => 16 : 030001,
GET_ERROR_CTRS => 16 : 080002,
READ_DATA_STANDARD => 16 : 080003,
RESET_ERROR_CTRS => 16 : 180002,
STANDBY => 16 : 190301,
STARTUP => 16 : 190302,
SELECT => 16 : 190303,
SECURE_RELEASE => 16 : 190401,
SET_RESET_HOLD => 16 : 190402);
```

Figure 3d—Representation specification for command enumeration type (ascending order of internal codes determines ordering in Figure 3a).

piled subunits (to disconnect the logical interface from the actual implementation). Except for the `read_data_standard` message, the message processing routines are relatively trivial. They perform their functions via the access routines provided in the `status_manager` package and the `error_counter_manager` package. Unsafe conversions are used, as necessary, to convert from Boolean arrays to the byte array of the out-going message buffer (and vice versa). A common routine, `valid_reply`, is used to set the appropriate return codes and transmit the reply by a hardware-implemented `start_io` routine.

The `read_data_standard` requests certain components from control blocks of the database. Rather than performing `UNSAFE_CONVERSION` on a record-component basis (which would require a large case statement to distinguish variant record parts), the conversion is done on a block-by-block basis, accessing the requested components by relative physical location in the block. (A table-lookup provides the necessary information.) Bypassing the strong typing of the logical representation of the control block database greatly simplified the procedure code (a 75 percent reduction in the number of statements required). The protection afforded by Ada's strong typing is superfluous in response transmission, since the output buffer is simply an array of bytes.

## SUMMARY AND CONCLUSIONS

As applied to a typical process control problem, the Model Controller Operating System, preliminary Ada sometimes helped and at other times hindered the program development process.

### *Hindrances*

The major deficiencies of preliminary Ada for real-time applications are the lack of a well-defined scheduler and the inadequacy of the mechanism for interrupt handling. The Ada language design team has acknowledged these problems, and, hopefully, will rectify them in the final language design. Otherwise, such functions will require a machine code implementation.

A related problem is synchronizing access to shared data in time-critical applications. Implementing mutual exclusion

using the rendezvous construct is awkward and inefficient as compared to other synchronization primitives such as spin locks.<sup>6</sup>

Another hindrance to program development is the required bottom-up textual presentation of information. This is exhibited by the restriction of no forward referencing in specifications. Although easier for compiler implementation, linear elaboration of declarations is not easier for either writers or readers of Ada programs. A textual presentation reflecting the top-down design process would be preferable.

Also detracting from the readability of Ada programs is the lack of explicit import lists of objects. Import lists are not required, yet without them program maintenance is hampered. To avoid the excessive burden on program developers, the import lists could be automatically generated by compilers or text editors.

### *Helps*

The major advantage of programming in Ada is the support provided by packages for encapsulation and information-hiding. The grouping of logically related data objects, types, and/or associated procedures greatly enhances the logical program structure. For instance, levels of protection of shared data objects may be implemented in the package body, concealing details from the users of the package.

Ada's separate compilation facility was used extensively to support modularization and enhance program structure. Separation of logical interface from physical implementation is a positive influence on program development.

Ada's strong typing is a definite plus. High level data definitions improve the readability of the code. In this regard, enumeration types are particularly useful.

A necessary companion to strong typing is the ability to escape it when a different view of the object is required, such as in decoding a message buffer. This is neatly provided by Ada's `UNSAFE_CONVERSION` function. `UNSAFE_CONVERSION` identifies those areas of the program where the safety checks of strong typing are temporarily suspended. Without this feature, a greater proportion of the program would have required machine code implementation. A related aid to systems programming in Ada is the coupling of logical to physical representations via the representation specification.

### *Issues in programming methodology*

During the design and coding of MCOS, some uncertainties about Ada were raised. They were eventually resolved as the authors gained experience with the language, and through consultations with various persons more closely connected with the Ada language development.\*

Issues identified during the MCOS exercise were: (1) the

\* In this regard, the authors would like to acknowledge John Barnes, Dennis Cornhill, Mark Davis, Robert Firth, John Goodenough, Oliver Roubine, and Peter Wegner.

interplay between traditional block structure and separately compiled modules, and how it affects program structure; (2) using visibility restrictions to advantage; (3) separate compilation of specification and implementation parts to reduce recompilation dependencies; (4) exception handling; and (5) dependency of logical representation on physical representation.

A programming methodology for Ada is required. A user's guide (an Ada cookbook) would facilitate program development. Due to the mixing of standard features with novel ones, the best Ada solution for a particular problem often cannot be ascertained. Current reference documentation<sup>1,2</sup> is inadequate.

### Conclusion

Without a doubt, systems programming is facilitated by using Ada, as compared to a full assembly language implementation. As with any high level language, a small pro-

portion (5-10 percent) of the program will require assembly language, either to maximize efficiency or to interface with hardware. Ada provides an interface to assembly code. Yet, due to the power of the language, machine-dependent code may be kept to a minimum.

### REFERENCES

1. "Preliminary Ada Reference Manual," *SIGPLAN Notices*, Vol. 14 (6), June 1979, Part A.
2. "Rationale for the Design of the Ada Programming Language," *SIGPLAN Notices*, Vol. 14 (6), June 1979, Part B.
3. Wegner, P., *Programming with Ada—An Introduction by Means of Graduated Examples*, New Jersey, Prentice-Hall, 1980.
4. Tannenbaum, A. S., "A Tutorial on ALGOL 68," *ACM Computing Surveys*, Vol. 8 (2), June 1978, pp. 155-190.
5. Wirth, N., *Algorithms & Data Structures = Programs*, New Jersey, Prentice-Hall, 1976.
6. Evans, A., Morgan, C., Roberts, E., and Clarke, E., "The Impact of Multiprocessor Technology on High-level Language Design," BBN Report No. 4188, September 1979.

# Computer aided heat penetration tests for the food canning industry

by PAUL SAGUES

University of California  
Berkeley, California  
and

The National Food Processors Association  
Berkeley, California

## INTRODUCTION

If food is to be preserved without refrigeration, drying, or curing of some form, then it must be thermally "processed." The result of processing is a commercially sterile product usually in a hermetically sealed container. Although food may be rendered sterile before it is packaged (the aseptic process), we will focus on the more common practice in which the product is sterilized after it is packaged. We will use the word "can" to represent both the container and the operation of sealing the container even though the words "glass jar" or "composite pouch" may be substituted for the container in most cases.

The antagonists in this scenario are various microorganisms. Spoilage organisms release gasses, swell containers, or produce putrefactive substances or toxins. One such organism, *Clostridium botulinum*, produces a toxin which is among the most deadly found in nature. This one species has spawned research through this century on questions relating to the ability of various microorganisms to survive thermal processes, and the ability of food processors to calculate acceptable levels of sterility.

## PROCESS CALCULATIONS

### Introduction

The evolution of process calculations is interesting but sufficiently relevant to merit only brief mention. Early workers<sup>1</sup> realized that the rate at which microorganisms died (lethal rate) is a function of the can temperature, time, the organism's "heartiness," and the nature of the product. The ideal process was described by a lethal rate curve (time vs lethal rate) whose area (total lethality) was unity. The strategy was modified in order that different organisms, products, and initial conditions could be represented. The modification involved introduction of an integral of lethality called the 'F value,' a measure of the time necessary to destroy a given number of organisms at a given temperature. (See Figure 1.)

In the first part of this century, Ball and others<sup>3</sup> recognized that a heating can may be described by a first order differential equation of the form

$$\frac{d\theta}{dt} = \frac{\theta}{\tau}$$

where  $\theta$  is temperature difference between the retort and container interior and  $\tau$  is a time constant which is determined by heat transfer parameters of convection, conduction, and effective specific heat of the product. This "time constant" is sometimes a function of time as in the case when a product heats by convection, absorbs the convective fluid, and then heats by conduction. In these cases, though, Ball chose a piecewise solution to avoid the process of integration. "[Determining the integral of lethality] by plotting the lethality curve and determining the area beneath it is a slow and tedious process."<sup>3</sup>

### Current methods

Three major methods are used to evaluate the integral of lethality. The first is the once tedious process of integration.<sup>2</sup> Although direct integration of lethal rate is well suited to computer data acquisition systems, we will—for reasons to be explained—direct our attention to the third method. The second major method of determining lethality<sup>4</sup> is the nomograph method which was well suited for use in the days of log tables.

The third method, Ball's solution to the determination of lethality (known as the formula method)<sup>5</sup> involves approximating the can heating curve as an exponential and plotting the log of temperature. The slope of the heating curve can be used in two ways. First, if the initial and process temperatures of the product are known and if the process time is known, then the F value may be computed. Second, the slope of the heating curve and the F value measure of the lethality can be used to determine the required process time.

From time and temperature records, processing authorities determine (usually post hoc) whether a given process

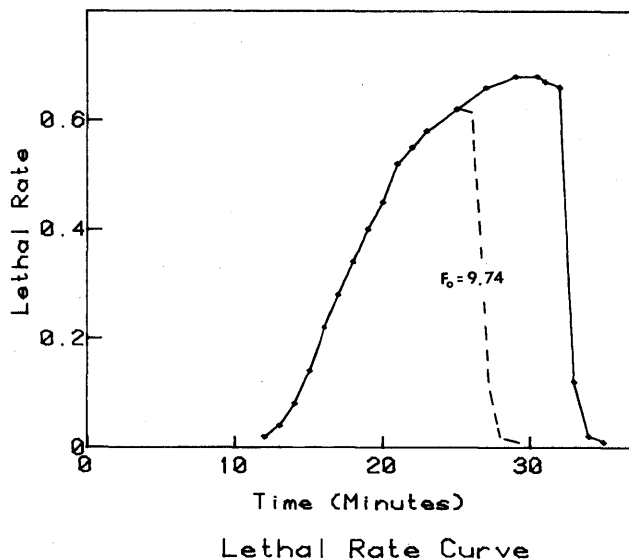


Figure 1—Typical lethal rate curve.

accumulated sufficient lethality to insure that the product is commercially sterile. The second use of the heating curve slope described above—applying lethality and temperature to determine process time—is used by processing authorities to establish the minimum initial temperature, process time, and process temperature for a given product. The food processor of low acid canned foods is legally obligated to meet the minimum calculated process parameters.

#### Heat penetration tests

The usefulness of any method of determining lethality is based upon the assumption that the can temperature may be determined. Theoreticians are tempted to model the system and determine the integral of the heating curve through simulation, but in the food industry the determination is empirical. The name given to the empirical method is a *heat penetration test*. A thermocouple is introduced through the can wall of the product in a location previously determined to be the slowest heating zone in the can. The product is then heated at process temperature until the can is within about 1C (2F) of retort temperature. The thermocouple's output is recorded on various devices such as a mechanical strip chart recorder. The collected data is reduced manually. The slope of the heating curve is found graphically, and a lag factor relating to establishment of process temperature is determined. Various computer programs determine process times from the hand entered data.<sup>6</sup> Heat penetration data is used by processing authorities to establish scheduled retort processes for low acid canned foods in the United States. For this reason, any effort to introduce current computer and control technology into the food canning industry should begin by carefully examining the heat penetration test in order that newly developed real-time process calculations will agree with proven empirical techniques.

## CURRENT RESEARCH

### *Motivating forces today*

Four factors have led us to believe that more effective thermal process monitoring and control is needed. First, energy conservation is not only a moral imperative but an economic necessity. Process steam requirements of the food processing industry are considerable. The Federal Government estimates that 4.8 percent of the total U.S. energy consumption is directed to the processing of food.<sup>7</sup> Second, the world market demands that productivity be increased and product waste be reduced. Third, producers and consumers strive for products which are processed more consistently. And fourth, impressive advances in microprocessor technology have at last offered realistic solutions to the problem of real-time data acquisition and control of thermal processes.

### *Goals of this research effort*

In light of the above four factors, we might ask, "Since technology exists to assemble hardware capable of real-time control, why not start building machines?" The answer has to do with the nature of government and industry. Both are fortunately conservative in accepting fundamental changes in an area so close to every person's well being.

We do not wish to re-invent food processing calculations. Rather, we would like to learn how to extend the capabilities of present mathematical methods of determining the integral of lethality. Our immediate goal in this effort, therefore, is to define what constitutes valid data acquisition by performing benchmark tests against accepted standard heat penetration methods. Because the majority of existing data is expressed in terms of heating curve slopes, we have chosen to begin by emulating data reduction by the formula method. Once confidence is gained that real-time mathematical solutions can adequately describe the proven graphical methods, our sights will shift to control of thermal processes. The tool around which our research effort is centered is a microprocessor based computer located in the process environment.

### *Design of the computer system*

#### Background

The National Food Processors Association (formerly the National Canners Association) Western Research Laboratory in Berkeley, California contains extensive bacteriological, chemical, and process research groups. Included within the facility is a pilot process plant which has simulators for most major thermal process methods. The facility has a Digital Equipment Corporation (DEC) PDP-11/34 minicomputer operating under RSX-11M multiuser real-time operating system.

Many considerations dictated the choice of a process level computer. The system was to be capable of communicating with the PDP-11/34 in order to have access to mass storage and data reduction and graphics programs. The system was to be operated by process engineers whose primary interest is food processing not computers. The system was to be sufficiently flexible that the user could define sampling parameters as well as display format. And finally, the process computer was to be sufficiently robust to stand up to the pilot plant environment. The choice for a process level computer was a DEC LSI-11/2 microcomputer which is located in the pilot plant and is connected by a 60m serial line to the PDP-11. The microcomputer is equipped with analog sampling hardware and has a cathode ray operator's terminal. The process computer has no mass storage and therefore no moving parts to foul in a hostile environment.

### Software

One rationale for choosing the LSI-11/2 is that it executes essentially the same instruction set as the PDP-11/34. All software may therefore be developed on the PDP-11/34 and down-line loaded to the LSI-11/2. In actual operation, the LSI-11/2 boots to a terminal emulator and the operator logs on the host computer. The operator then runs a supervisor program in the host which signals the LSI-11/2 to accept a task image. A task image contains an operating system, real-time sampling routines, and calculation program.

### Process computer operating system

The LSI-11/2 operating system was written in house in assembly language to provide (1) reliable message transmission and reception with the host, (2) a memory mapped video for the display of real-time data in a format determined by the user, (3) operator input/output capability with either the LSI-11/2 or host, and (4) provision for use of a high level language to perform real time calculations. Error checked packets are sent and received by the LSI-11/2 for the purpose of transmitting data, formatting the memory mapped video, passing sample rate parameters, and interacting with the user. The LSI-11/2 operating system resides in a library in the PDP-11/34 and is built into a task before being down loaded to the LSI-11/2. The read only memories in the LSI-11/2, therefore, need not be updated if the operating system or programs are changed.

### Process computer capabilities

Presently, Fortran is the high level language operating in the LSI-11/2. The function of the Fortran program is depicted in a state transition diagram<sup>8</sup> in Figure 2. Fortran's ability to compute and manipulate multidimensional arrays with reasonable ease is exploited as it linearizes sampled data, organizes packets bound for the host, and structures infor-

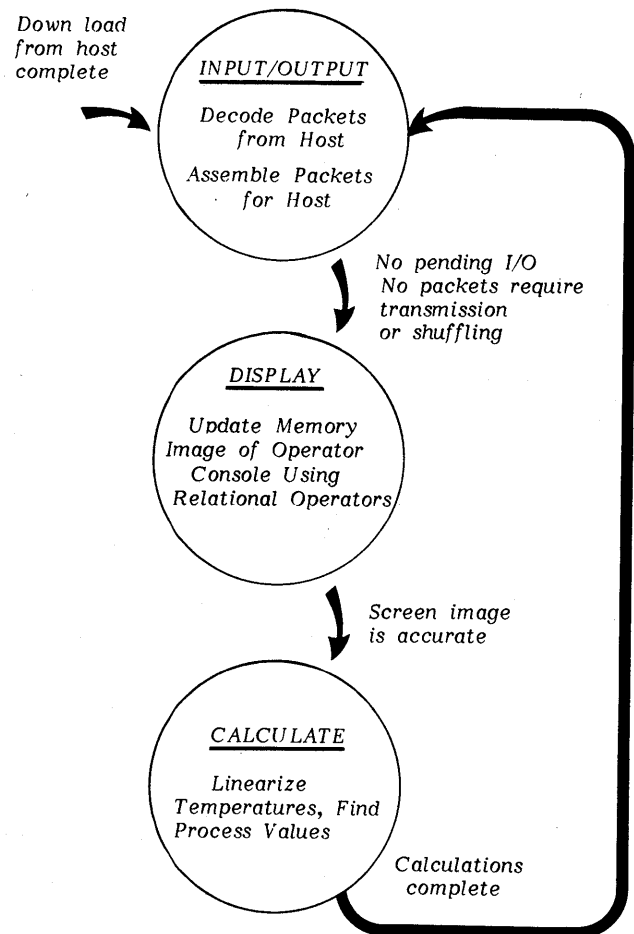


Figure 2-State transition diagram of process level computer main program.

mation for the user. Process calculations described by Ball in 1923 as tedious are performed by Fortran routines.

A central construct of the microcomputer operating system is a dynamically changing section of memory which easily can be mapped to most cathode ray terminals. The memory map is defined as a set of operations on several arrays managed by the high level language. These arrays include real-time values, thus through the use of simple, interactive commands from a "display language," the user can build a video image comprising strings or real-time values in such a manner as to tailor the system to his or her liking. The display language has been an important consideration since we wish the system to evolve into a tool which is of a form we cannot presently define precisely.

In our continuing effort to make this system simple to use and yet flexible, we have incorporated a command file structure which allows each user to maintain a file of non-procedural commands. When this file is executed, the process computer is configured to perform a specific test for a specific type of product. New display formats may be built and sampling strategies tested by operators who are not computer professionals.

### Sampling system

Industry standard type "T" thermocouple pairs are our thermal-electric transducers. One junction of the pair is introduced into the process vessel or can while the second (reference) junction is maintained at 0C in an ice point reference (Omega TRC). The low level signals are multiplexed by flying capacitor isolators to a differential amplifier of gain 1000. A successive approximation analog to digital converter produces a 12 bit digital representation of the voltage.

### Precision and accuracy

The mercury thermometer remains the secondary standard in the food industry. To obtain our goal of reproducing results obtained with existing heat penetration methods, we must know that at least the precision of our sampling system is acceptable to processing authorities. Our preliminary findings appear to confirm Roop and Badenhop's results<sup>9</sup> that accuracy is limited by precision, and precision is 0.094C (0:17F). Extensive stability testing will be required to substantiate these claims, but initial findings indicate that our 12 bit conversion scheme will yield sufficient precision for thermal processing of food.

### Sampling strategy

Two major goals were defined during the design of the data acquisition system. First, a sufficient number of data points were to be recorded—in a manner readable both by a computer and a human—that the log of a heat penetration test may be plotted manually and yield results at least as precise as allowed by the current manual method. Second, no more data was to be recorded than is necessary to represent a process to the limit of the precision of the instruments used in the test.

In order to meet these goals, we have designed a data compression sampling routine whose sampling parameters may be tailored by the user. Our system nominally samples at ten millisecond intervals and applies criteria based upon both rate of change of the input signal and the precision with which the accumulated lethality may be represented. The sampling technique appears well suited to heat penetration tests which may be as short as several minutes or as long as several hours.

### Preliminary results

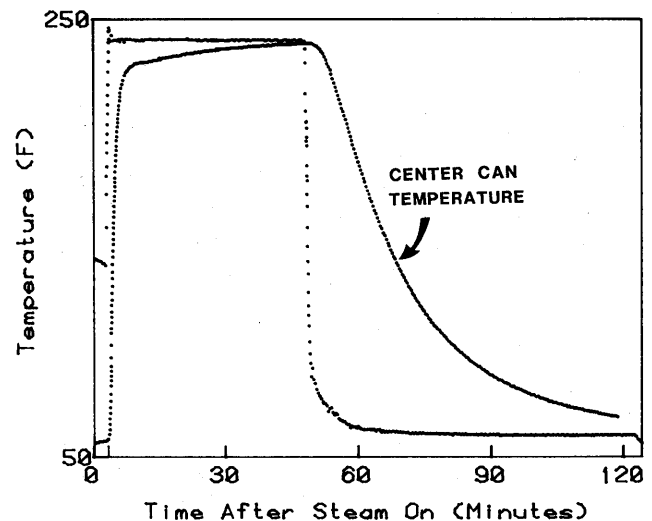
Our process computer system became operational a short time ago and at this writing only a few heat penetration tests have been run. We are very pleased with the performance of our system in these tests. Initial analysis of the results indicates that the information content of the data is significantly higher than the currently used method. Rigorous side-by-side tests will occupy our time in the near future.

In one test we placed 225g (8oz) of dry, white beans in

each of two 16 ounce cans and filled the cans with water. The cans were fitted with an Ecklund type "T" thermocouple located 1.9cm (0.75in) above the bottom of the can. The cans were sealed and placed vertically in a still retort. The cans were processed at 115.69C (240.25F) for 44 minutes. Three signals were recorded. The thermocouples of both cans and a retort temperature thermocouple were logged for 120 minutes. Data was transmitted to the PDP-11/34 and stored on a flexible disk. About twelve hundred data points were recorded in our test. Figure 3 shows the relation between retort temperature and can temperature. (Graphs presented were drawn using the flexible disk data files by the program "GD" written by Professor D. M. Auslander of the University of California, Berkeley).

We chose to process dry beans out of curiosity. As mentioned above, a product which heats by convection and then changes such that the convective fluid is no longer present exhibits a "broken" heating curve. The beans absorb the water and heat by conduction which is a slower mode of heat transfer in this case. This type of curve is not a favorite among those reducing data since the location of the transition is often ambiguous. We decided to begin with a mixed mode product and hoped to see a broken curve when the data was plotted on a logarithmic scale.

The can temperature data of Figure 3 is plotted in Figure 4 on axes scaled in industry standard form for a heat penetration test. The resulting plot indicates that we can resolve the broken curve transition area with precision which exceeds that obtained using current techniques. We have seen far less ideal curves produced by computer simulation which indicates that our next goal of interpreting data in real time may not pose problems which tax our analytical ability. Even the cooling curve—the portion of Figure 4 to the right of the maximum—is noise free and continuous. Cooling curves are often ignored in process calculations although the area beneath them contributes to the integral of lethality. Our pre-



Heat Penetration Test  
Retort and Can Temperature

Figure 3-Heat penetration test results.

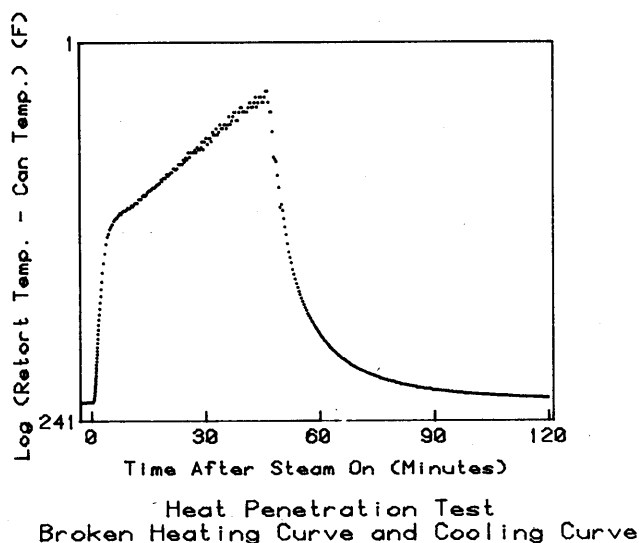


Figure 4-Heat penetration test results: standard form.

liminary results strongly indicate that we can fulfill our research goals.

## CONCLUSION

American food processors are to be commended for their impressive record of providing pathogen-free products to the consumer.<sup>10</sup> Much of the credit must be given to those who have developed the empirical methods of predicting what constitutes an acceptable error bound for the probability of destroying a population of microorganisms. But technology has progressed to the point where processors will soon be able to increase their quality assurance and at the same time reduce thermal energy demand, increase productivity, and improve the nutritive properties of their product.

Our goal is to follow in the path of the empiricists by emulating their techniques. Such a tack will allow us to build tools which may then evolve with advances in technology. These tools will not be built by computer scientists alone, or by food processors alone, or by control engineers alone.

Success will come through careful cooperation. Our research is in an early stage. The majority of our effort thus far has been spent defining and designing what will become an everyday tool. We look forward to an extended testing phase. But feasibility of applying current technology to the food processing industry is already evident.

## ACKNOWLEDGEMENTS

The author wishes to thank the staff of the National Food Processors Association for their patience and understanding. Specifically, Doug Sasseen, Larry Lewis, Rick Kimball, and Jay Unverferth have done all in their power to make this project a success.

The guidance and inspiration of Professor D. M. Auslander is especially appreciated.

## BIBLIOGRAPHY

- 1 Prescott, S. C. and Underwood, W. L. "Microorganisms and Sterilizing Processes in the Canning Industry," *Technol. Quarterly*, Vol. 10, No. 1 (1897).
- 2 Bigelow, W. D., Bohart, G. S., Richardson, A. C., and Ball, C. O., "Heat Penetration in Processing Canned Foods," National Canners Association, Bulletin 16L (1920).
- 3 Ball, C. O., "Thermal Process Time for Canned Food," *Bulletin of National Research Council*, Vol. 7, Part 1, No. 37 (October 1923).
- 4 National Canners Association (compiled by), *Laboratory Manual for Food Canners and Processors*, Vol. 1. Westport, Conn., AVI Publishing (1968).
- 5 Ball, C. O. and Olson, F. C., *Sterilization in Food Technology*, New York, McGraw-Hill (1957).
- 6 Sasseen, D. M., "Interactive Plotting of Heat Penetration Test Data," National Food Processors Association (1979) (in press).
- 7 FEA, Energy use in the Food System, Office of Industrial Programs, Federal Energy Administration, U.S. Government Printing Office, Washington, D.C. (1976).
- 8 Auslander, D. M., Dornfeld, D., and Sagues, P., "Software for Microprocessor Control of Mechanical Equipment," *Proceedings 1979 Joint Automatic Control Conference*, Denver, Colorado, June 1979.
- 9 Roop, Richard A., and Badenhop, Arthur F., "A Computer-Thermocouple Interfacing System for Time-Temperature Data Collection for Thermal Food Processes," *Journal of Food Processing Engineering*, Vol. 4, No. 2 (1979-1980).
- 10 U.S. Department of Health, Education, and Welfare, Public Health Service, "Botulism in the U.S. 1899-1973," DHEW Publication No. (CDC)74-8279 (1974).





# A cross-impact simulation forecast of the data processing industry

by PAUL HERBERT ROSENTHAL

*Gottfried Consultants Inc.*  
Los Angeles, California

This paper describes the application of the USC Center of Future Research's Cross-Impact Model to the simulation of the evolution of an emerging industry. The Data Processing Industry (DP) and its emerging Network and Decentralized Systems sub-industries are used to illustrate the application of the model. This application of the USC cross-impact forecasting model to the simulation of the evolution of an emerging industry and its component parts will constitute its first micro-economic application.

During recent data processing forecasting projects, it became apparent that a model describing the behavior of complex emerging industries was needed to facilitate forecasting the structure of such industries as data processing, calculators, solar energy and copiers. The author, therefore, embarked on applying the methodology of the Interactive Cross-Impact Model to demand analysis of such technology-based emerging industries using the DP industry as a case study.

The objective of this paper is, therefore, to present a methodology for forecasting and analyzing the effect of technology innovations and policy interventions on the growth and structure of an emerging industry, such as Data Processing. The Network Information Services and the Decentralized Systems sub-industries were chosen as illustrative of the class of technology-based emerging industries, since they are currently at a crossroads in their development due to the emergence of new competing and enabling communications and mini-computer technologies and services. More detail on the model and methods can be found in *A Cross-Impact Simulation of an Emerging Industry: A Case Study of Data Processing* (Rosenthal, 1979).

## EMERGING INDUSTRY STRUCTURE

The model described defines three levels of emerging industries: the independent new industry, the primary industry segment—offering new services, and the subindustry segment—offering new production methodology for supplying an existing service.

### *Independent industry*

The independent or total industry level is illustrated by the Data Processing Industry. It would have been considered by Lynn (1966) a "new" industry and its evolution would be measured by its total sales growth or its change in GNP percentage.

### *Primary industry segments*

The primary industry segment level is illustrated by Centralized Data Processing Services, Decentralized Data Processing Services, and the Networked Services Industries. This level is characterized by product differentiation. For example, to the user these industries offer different services and products for use in different applications. For the technologist, these industries are simply varying delivery vehicles using varying mixtures of the same or different technologies.

### *Subindustry segments*

The subindustries level is illustrated by the Network Information Services (NIS) Industry and its competitive, internally provided Dedicated Networked Systems Industry. This level is characterized by multiple delivery or technological approaches to providing the same product.

It is this level that much of the technological change literature approaches. For example, Gold, Pierce, and Rosegger (1975), in their paper on "Diffusion of Major Technological Innovations," measure the proportion of total output accounted for by fourteen major production process innovations in the Iron and Steel Industry.

Both this level and the primary industry segment level are measured in the model by share of market and by penetration rate (rate of change of market share). This approach cancels out the effects of total industry growth.

**THE CROSS-IMPACT FORECASTING PROCESS**

The cross-impact forecasting system consists of: a generic simulation system that operates in either scenario or statistical mode, a user-provided application model, and a data base of input parameters and output variables.

*Data base contents*

The data base consists of an input parameter file defined and completed by the user and FORTRAN COMMON tables defined by the generic cross-impact system.

The input parameter file created by the user contains the following information for the emerging industry model:

- a. Event Probabilities—cumulative innovation event probabilities. These probabilities are normally derived through interviews with experts.
- b. Cost Performance Trends—nominal cost/performance indexes derived from technology and technology diffusion equations.
- c. Event/Event Impacts—event-on-event odds multipliers and delay/decay time periods that determine a period of applicability. These values are also derived from experts' interviews.
- d. Event/Trend Impacts—event-on-trend multipliers and delay/decay time periods that determine the length of time an innovation takes to impact product cost/performance. These values are determined from forecasts and technology diffusion equations.
- e. Elasticity/Substitutability Coefficients—a series of parameters determining the impact level of cost/performance on total industry growth and industry segment market shares. Their values are derived by tuning the model to historic data or to short-term forecasts.
- f. Initial Industry Structure—A series of initial values for the current size of the industry. These values are derived from historic data.
- g. Names—names are given for events, trends, and output variables for clarity of output reports.

The output variable tables are used during computations to store intermediate values and are formatted and printed on the output reports. Data is maintained for the three levels of emerging industries: independent total industry, primary industry segments, and subindustry segments. The data maintained for each level includes:

- a. Current Dollar Sales—dollar sales in current dollars for each time period of the simulation.
- b. Real Dollar Sales—dollar sales for each time period in real dollars using the initial year of simulation as the base year.
- c. Growth Rate—percent growth rate for each time period of real dollar sales.
- d. Market Share—market share of segment for each time period.

- e. Technology Index—the value for each time period of its cost/performance trend associated with the segment. This value is the nominal value adjusted by the impact of innovation (event) occurrences.
- f. Penetration Rate—the rate of change of market share for each time period of the segment.

*Scenario mode simulation*

Figure 1 outlines the structure of scenario mode simulation. Each interaction with the forecasting system defines and performs a single simulation forecasting run covering the time period of the forecast. The process consists of the following five steps.

**Define industry model**

This step involves the creation of the input data base and an initial interactive intervention defining the simulation period and the initial random number seed.

**Initialize industry variables**

This step consists of an emerging industry FORTRAN application program that moves initial values from the input file to the variable tables.

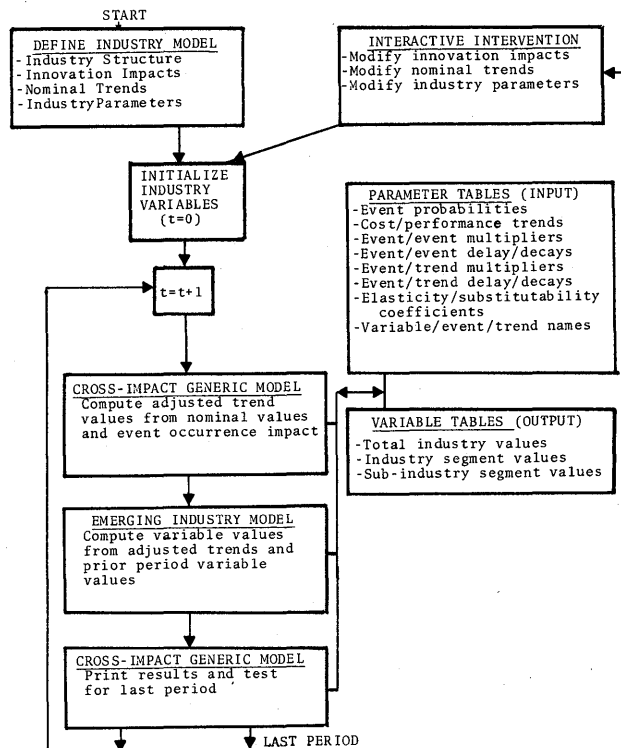


Figure 1—Scenario mode simulation.

### Cross-impact generic model

This step is performed by a large and complex generic cross-impact FORTRAN program. A Monte Carlo simulation methodology is used to determine event occurrence based on accumulative event occurrence probabilities. When an event (an innovation or policy intervention) occurs, odds multipliers are used to modify other event probabilities and multipliers are used to modify cost/performance trends. The impact of an event occurrence can be spread over several periods through its use of delay and decay coefficients. A later step is also performed by the same program consisting of output printing of scenario results and a test for last time period.

### Emerging industry model

This step is performed by an emerging industry FORTRAN application program that computes variable values based on adjusted trends and prior period variable values.

### Interactive intervention

This generic cross-impact FORTRAN program allows interactive modification of probabilities, input trends, and non-structural emerging industry parameters such as elasticities and substitutability coefficients. The program will not change the number of events, trends, or variables.

### Statistics mode simulation

Figure 2 outlines the structure of statistics mode simulation. Each interaction with the forecasting system causes multiple scenario simulations to be run, each utilizing a different set of random numbers to generate event occurrences.

At the completion of each scenario, sums and sums of square are accumulated for use in computing means and standard deviations of trend values. Means are also computed for event occurrence frequency and segment variable values including current dollars, real dollars, growth rate, market share, technology index, and penetration rate.

### INDUSTRY STRUCTURE

A straightforward, two-level tree emerging industry structure is utilized as shown in Figure 3.

The total Industry is defined as the sum of its segments each characterized by highly differentiated product or service types. The share of market of each industry segment is assumed to be based primarily on the cost/performance of their products with moderate substitutability over time.

The simulation model is used to trace the sudden emergence of a new subindustry product or service within one of the industry segments because of substantially improved product cost/performance. The parent segment then ex-

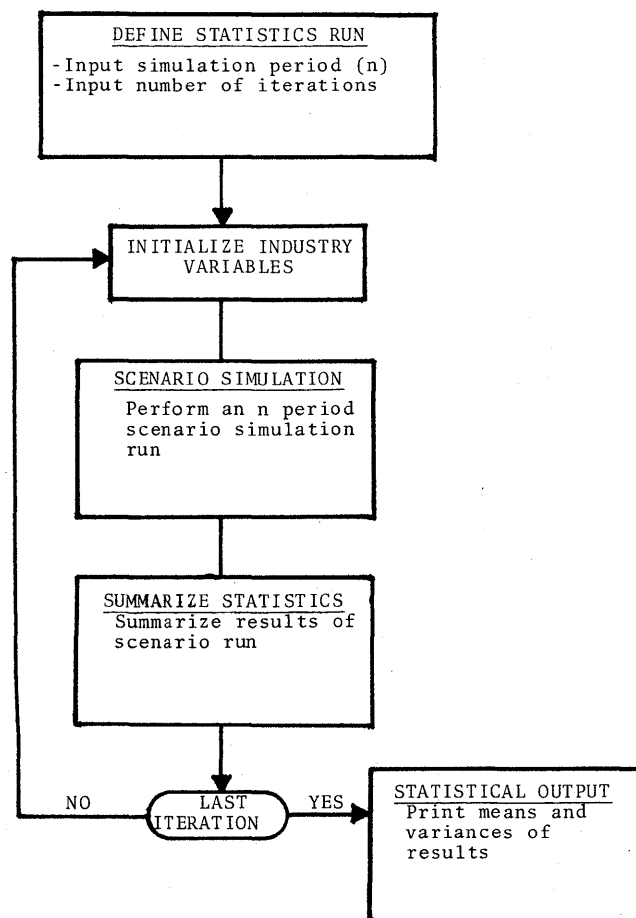


Figure 2—Statistics mode simulation.

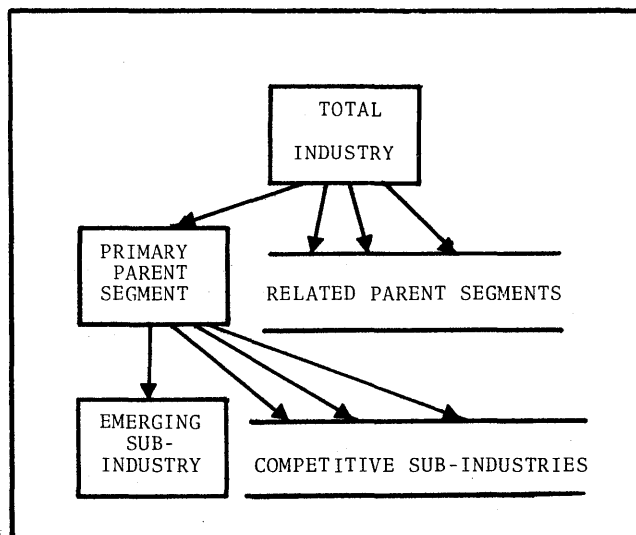


Figure 3—Definition of industry structure.

pands, creating expanded research and development activities in the related industry segments, which may then produce innovations increasing the cost/performance of related products.

As an example of this type of industry evolution, the data processing industry structure in Figure 4 will be used.

Note that the Data Processing Industry is defined not by sales of components or services (such as minicomputers, printers, leased lines, or software packages), but instead by end user deliverable systems which meet specific information processing needs. Because of this structural approach, technological innovations in components and services, such as micro-electronics and communication services, will differentially impact all industry segments with one segment often receiving the primary benefit.

**MARKET STRUCTURE**

The emerging industry application model was incorporated into the USC generic cross-impact model program, and the elasticities tuned to fit 1975 through 1983 traditionally derived Data Processing Industry forecasts.

The equation used for total industry demand was:

$$S_t = S_{t=0}(1+r)^t(1+\eta\sqrt{T_t - T_{t=0}})$$

- where  $S_t$  = total industry constant dollar sales at period  $t$ .
- $r$  = basic annual real growth rate of the economy (a value of .035 was used for the data processing industry simulations).
- $T$  = weighted mean technology index of cost/performance trends.
- $\eta$  = technology/demand elasticity coefficient (simulations used a value of .015 for the data processing industry.)

The demand equation can be derived as the product of three factors: initial industry size, real GNP correlate, and emerging technology correlate.

$$S_t(\text{constant dollar sales}) = S_0(\text{Initial Industry Size}) \times f(\text{real GNP}) \times g(\text{cost/performance})$$

A long-term view of GNP growth is used for the second term of the demand equation.

$$f(\text{real GNP}) = (1+r)^t$$

Total Industry	Data Processing Industry
1. Parent Segment	Network Systems
A. Emerging Sub-Industry	Network Information Services
B. Competitive Sub-Industry	Dedicated Networked Systems
2. Related Segment	Centralized Systems
3. Related Segment	Decentralized Systems

Figure 4—Illustrative market structure.

The traditionally high correlation between GNP and industry demand is widely used in forecasting (Spencer and Siegelman, 1964, page 150). Its use in the demand equation accounts for growth due to an expanding total economy.

The emerging technology term is derived from the simplest functional form of a declining marginal utility curve (Tintner, 1965, page 54).

$$\frac{dS}{dT} = aT^{-b} \text{ where } 0 < b < 1$$

Solving the differential equation and substituting for initial conditions at  $t=0$ , gives the basic production/demand function used.

$$S = nT^d + 1 \text{ where } d = -b + 1 \text{ implying } 0 < d < 1$$

The last step in the derivation is the determination of the exponent ( $d$ ) of the technology term  $T$ . The mid-point of the range for the exponent (.5) was used as an initial value, as did Wahi (1972). Based on the close fit of the derivative of the simulation result curves to the calibration data, no later adjustment was made.

The equation used for determining market share at both the primary industry level (product differentiation level) and at the sub-industry segment level (technology or delivery system differentiation level) was:

$$M_{i,t} = M_{i,t-1} \left( \emptyset + (1+\emptyset) \frac{nT_{i,t}}{\sum_i T_{i,t}} \right)$$

where  $M_{i,t}$  = market share of  $i$ th industry segment during period  $t$ .

- $\emptyset$  = substitutability coefficient
- if  $\emptyset = 0$  there is complete substitutability, no product differentiation. If  $\emptyset = 1$  there is no substitutability, isolated markets. (The data processing industry simulation used values of .25 and .15 for the primary and subindustry levels.)

The industry segments used were:

- $i=1$  All Network Services (sum of  $i=4$  and  $5$ )
- $i=2$  Centralized Services
- $i=3$  Decentralized Services
- $i=4$  Network Information Services
- $i=5$  Dedicated Network Services

These econometric demand functions are similar in format to the equations used in numerous management simulations games; see, for example, the Yale University game (Shubik, 1964) and the IBM game (Wahi, 1972).

The simulation model was used to produce a fifteen-year forecasting run of the data processing industry from 1975 through 1990. Three types of simulation runs were performed.

*Nominal scenario*

A nominal scenario was run utilizing existing variable inter-relationships, nominal trends, and excluded the impact

of future events. This run operated the simulator model in the same deterministic format as traditional short- and medium-term simulation models and excluded the long-range forecasting cross-impact elements.

*Stochastic scenario*

A stochastic scenario was run utilizing cross-impact probabilities and impacts. This run used Monte Carlo methods to approximate the impact of occurrence of abrupt events and changes.

*Alternate policy scenarios*

Several alternate stochastic scenarios were run incorporating the specialized policy event, "White Collar Unionization Expansion." These runs demonstrated the methodology for using the simulator model for studying the impact of potential policy interventions.

The deterministic forecast tracked a composite forecast combining such diverse groups as: Frost and Sullivan (*ComputerWorld*, January, 1977), INFORUM research project (Almon et al., 1974) and Quantum Sciences Corporation (MAPTEK, 1975). The stochastic simulation forecasts tracked the International Data Corporation (IDC) forecasts (*Fortune*, March, 1977).

OPINION SURVEY INPUT DATA

A survey was performed of several 'experts' in the data processing industry who specialize in technology forecasting. The experts were requested to supply the following subjective estimates:

- a. Cumulative probability of innovation/regulation event occurrences (assuming non-occurrence of the other events).
- b. Event-on-event cross-impact coefficients.
- c. Event-on-trend impact coefficients and delay/decay periods.

Figure 5, "Event-on-Event Impact Structure," shows the

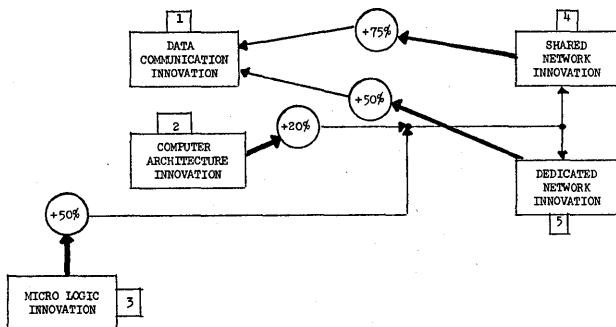


Figure 5—Event-on-event impact structure.

structure of the event-on-event odds modifiers derived from the opinion survey and used in the Data Processing Industry simulation.

The impact of the occurrence of an innovation or policy event on the value of a cost/performance trend is multiplicative in nature. The results of the opinion survey produced the structure shown in Figure 6.

RESULTS OF THE SIMULATIONS

The model produces annual real dollar and current dollar figures for each segment of the emerging industry. The following table (Table I) presents the 1990 forecasted constant dollar results for the nominal and stochastic scenarios.

TABLE I.—Forecasted 1990 Data Processing Industry Sales (in billions of 1975 dollars)

Type Service	Nominal Scenario	Stochastic Scenario
Centralized	24	23
Decentralized	16	16
NIS	5	28
Dedicated Network	10	12
Total DP Industry	\$55	\$79

The introduction of the innovation occurrences significantly expanded the market while changing its structure.

The detailed results of the simulation forecasts are shown in Figures 7, 8, and 9. These charts include stochastic simulation results, nominal simulation results, calibration forecasts, and the IDC fortune forecast. The charts indicate that IDC appears to include some level of new technological innovations impact in their five-year and over forecasts.

The Primary Industry Segment Forecast Chart illustrates an unexpected trend resulting from the cross-impact patterns. By 1990, the growth of the data communication-oriented industry segments can be expected to reduce centralized and decentralized stochastic forecasts to the same level or less than the nominal forecasts.

TABLE II.—Alternate Scenario Results (assumed policy intervention)

	1990 Statistical Results	1990 Alternate Results	Percent Reduction
Total DP Industry	\$214M	\$194M	-9.3%
Networked Systems	106M	97M	-8.5%
Centralized Systems	64M	56M	-12.5%
Decentralized Systems	45M	41M	-8.9%
N.I.S. Segment	78M	68M	-12.8%
Dedicated Segment	28M	29M	+3.6%

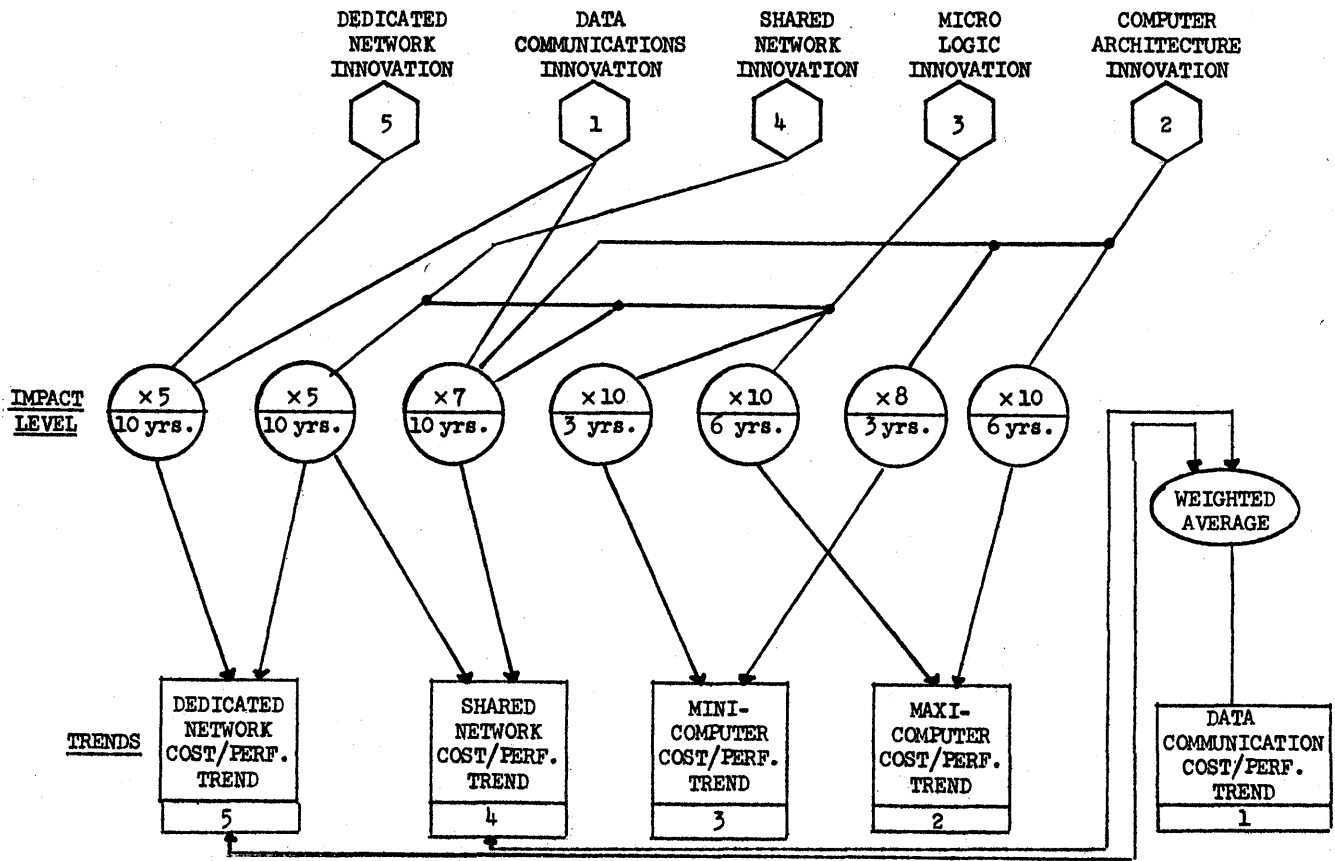


Figure 6—Event-on-trend impact structure.

TABLE III.—Total DP Industry Forecasts (in 1975 billions of dollars)

Model Results	1980		1983	
	\$	ERROR	\$	ERROR
Nominal Scenario	50.0	1%	69.9	1% of compromise
Stochastic Scenario	53.9	6%	78.8	1% of IDC
<u>Validation Data</u>				
Composite Forecasts	50.5		69.5 (11.2% growth rate)	
IDC Forecasts	>50		78 (15.6% growth rate)	

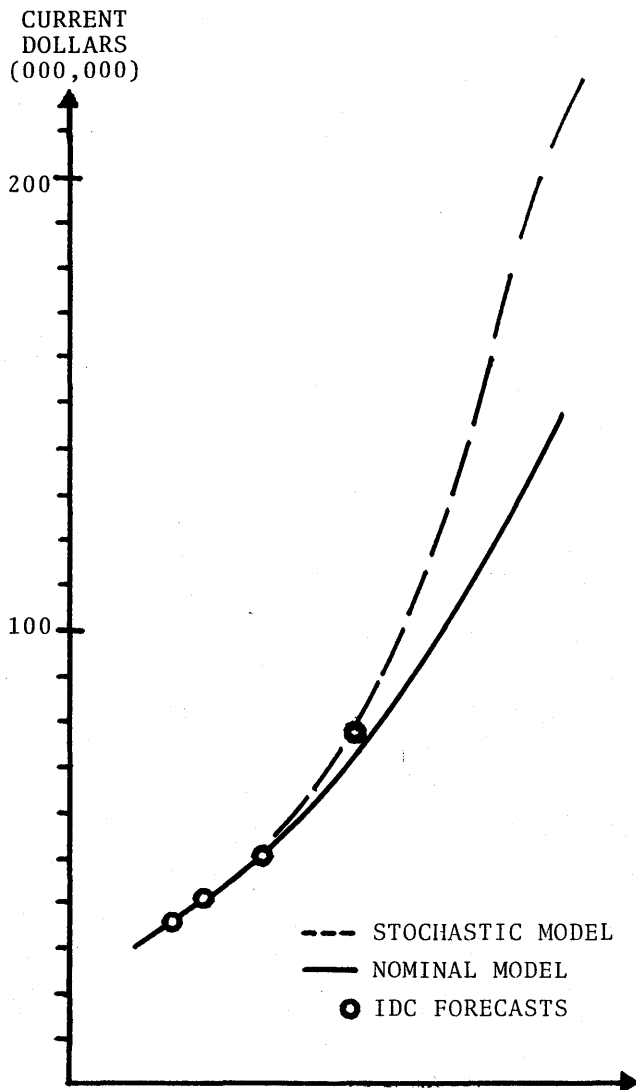


Figure 7—Total data processing industry forecasts.

#### ALTERNATE FUTURES FORECASTS

Researchers and planners are often interested in a forecasted industry profile based on the occurrence of selected exogenous events. Such simulations use the impact coefficients of technical and policy event occurrences on cost/performance trends. Typical exogenous events include governmental policy interventions or union activism. An illustrative simulation was performed that used "White Collar Unionization Expansion" as an exogenous event.

The alternate simulation modified industry segment growth in Table II.

All segments and industries other than Dedicated Networked Systems were reduced by approximately 10 percent, losing almost one-third of their gain over the Nominal Simulation. Since the Dedicated Segment lost sales late in the simulation when stochastic event occurrence was intro-

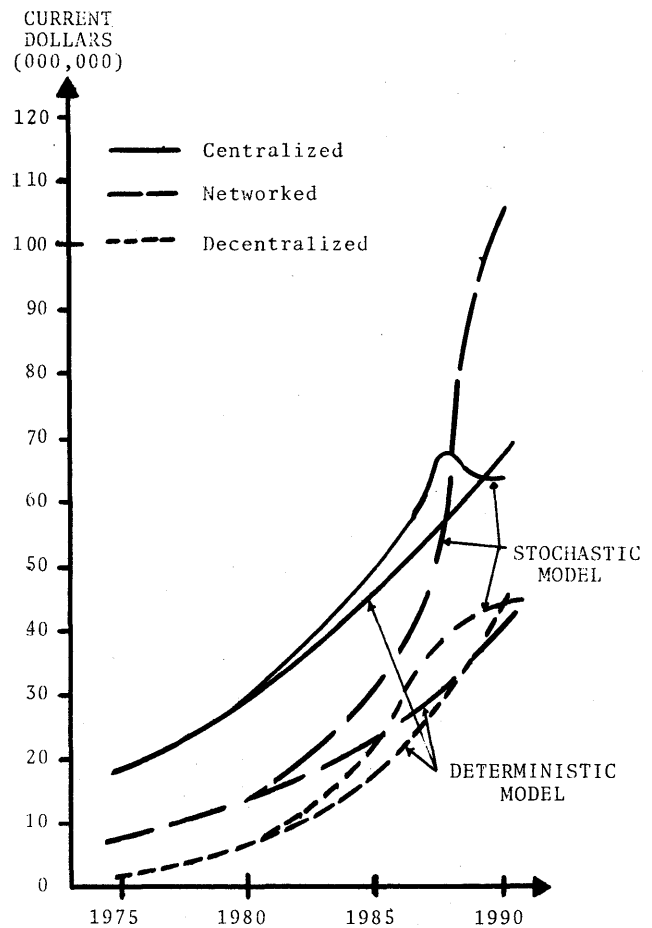


Figure 8—Primary segment forecasts.

duced, the reduction in their impact due to the policy variable would be expected to give the gain shown in the simulation.

#### ACCURACY OF FORECASTS

The forecasts produced by simulation are compared to the independently obtained composite forecasts in Table III.

#### IMPLICATIONS OF DP INDUSTRY FORECAST

The results of the DP Industry forecast are summarized in Table IV and show the forecasted growth rates of each industry segment. These growth rates were derived from the constant dollar expenditure forecasts summarized in Table V.

The figures are based on the results of the stochastic simulation. The expenditure data has been adjusted to 1980 dollars from the 1975 dollar figures used in the prior calculations.



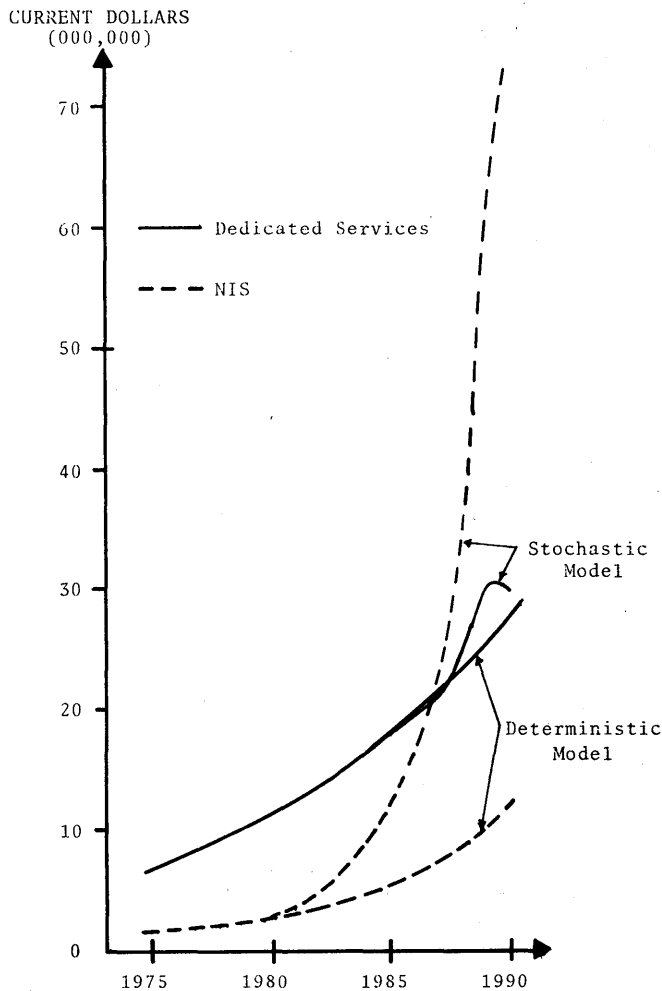


Figure 9—Data communication sub-industry segment forecasts.

### Trend summary

The results shown in Tables IV and V indicate that the 1980's will be a period of rapid growth for the data processing industry with real growth running at least twice that of the total economy. The early 1980's will see rapid growth for decentralized minicomputer-based systems as well as for

TABLE IV.—Forecasted Growth Rates

	Annual Compounded Real	
	Dollar Growth Rates	
	1980-1985	1985-1990
Total DP Industry	6-1/2%	8%
Centralized Services	2%	-2%
Decentralized Services	18%	7%
Networked Services	8%	20%
NIS	27%	32%
Dedicated	0%	3%

TABLE V.—Forecasted Constant Dollar Expenditures (in billions of 1980 dollars)

	Annual Real Dollar Internal and External Expenditures		
	1980	1985	1990
Total DP Industry	\$54	\$74	\$109
Centralized Services	32	36	32
Decentralized Services	7	16	23
Networked Services	15	22	54
NIS	3	10	40
Dedicated	12	12	14

NIS services, while the late 1980's will see a continuation of the NIS growth but a significant slowing in the growth of decentralized systems.

While real dollar sales of the DP Industry will grow at 6 percent to 8 percent during the decade, its largest and oldest segment, centralized large-scale computer-based systems, will remain constant in size during that period. Approximately two-thirds of DP growth will be accounted for by NIS expansion with the remaining one-third accounted for by expanding decentralized systems.

These DP Industry forecasts, although surprise-free in the sense that no unusual trends developed, did generate some unexpected results and contributed to an understanding of the dynamics of DP Industry growth. The forecasts indicate that the growth of network services during the 1970's through dedicated networks will, during the 1980's, switch to shared network utilization via NIS-type services. The next decade, therefore, should see a rapid growth of Value Added Network carriers that offer shared data communication services, permitting electronic mail and data base access.

The forecast also unexpectedly showed a slowdown in the erosion of centralized data processing services. The 1970's were a decade of rapid transfer of centralized batch applications to decentralized and networked services. This erosion will slow, and centralized services will grow slightly during the 1980's as expected improvements in large-scale computer productivity make integrated management information systems practical for most firms.

Also unexpectedly, minicomputer-based decentralized services growth will slow to the industry average by the mid-1980's. This indicates that increasingly the minicomputer vendors must, within the next several years, look to network-oriented products (not computational products) if they wish to maintain their rapid growth.

### Impact on management

The primary impact of the continued expansion of decentralized information systems over the next five years will be the ability of small and satellite organizations to automate such day-to-day applications as: order entry, inventory con-

trol, billing, and office automation. The productivity of the small firm or office will be improved and the current trend toward managerial decentralization should continue into the mid-1980's.

The primary impact of the rapid and continuing growth of NIS through the 1980's will be the growth of integrated systems linking vendors, suppliers, remote offices, and homes (Diebold, 1977). This trend should start to reverse the managerial decentralization trend of the mid-1980's as corporate management and staff link their decentralized systems through NIS. They will then demand compatibility and simultaneously achieve the capability for day-to-day monitoring and control of remote operations. In summary, during the 1980's, electronic communication and storage of information will be less expensive than paper-based systems, expanding the pace of white-collar automation. This direction is well summarized by Charles P. Lecht (1977, page 178).

While it may yet be possible to argue that if you destroyed all the computers in existence today it wouldn't *seriously* affect your life, it seems equally obvious that by the early 1980's we would be drowning in paperwork. Progress would necessarily retreat without the powerful computer systems and networks upon which a service-oriented society is—we would say inescapably—dependent.

The expanding computer/communication environment forecasted will, therefore, expand the impact of data processing on the economy. Management must plan for this expanding role of Information Automation, and assure its profitable use.

## BIBLIOGRAPHY

- AFIPS, *Information Processing in the United States: A Quantitative Summary*. Montvale, NJ: AFIPS Press, 1976.
- Almon, C., Buckler, M. B., Horwitz, L. M., and Reibold, T. C., 1985: *Interindustry Forecasts of the American Economy*, Lexington, Mass.: D. C. Heath and Company, 1974.
- Alter, S. *The Computational Mathematics of Time-Dependent Cross-Impact Modeling*. M26, Center for Futures Research, University of Southern California, October 1976.
- Alter, S. "The Evaluation of Generic Cross-Impact Models," *Futures* 2, April 1979, pp. 132-149.
- Armstrong, J. S. *Long-Range Forecasting: from Crystal Ball to Computer*, New York: John Wiley, & Sons, 1978.
- Chow, Gregory C. "Technological Change and the Demand for Computers," *The American Economic Review*. 57 (5), December 1967, pp. 1117-1130.
- Diebold Research Program, *The Implications of Converging Computer and Communications Technology*. M40, New York: The Diebold Group, Inc., August 1977.
- Dolotta, Bernstein, Dickson, France, Rosenblatt, Smith, and Steel. *Data Processing in 1980-1985*, New York: John Wiley & Sons, 1976.
- Duval, A., Fontela, E., and Gabus, A., *Cross-Impact—A Handbook on Concepts and Applications*, Geneva: Battelle, 1974.
- Frost and Sullivan, Inc., "Time-sharing Seen Losing Ground as Service Choice," *ComputerWorld* XI (3), January 17, 1977, p. 47.
- Gold, B., Pierce, W. S., and Rosegger, G., "Diffusion of Major Technological Innovations," in B. Gold (Ed.). *Technological Change: Economics, Management, and Environment*, New York: Pergamon Press, 1975.
- International Data Corporation, "Distributed Processing/Data Communications," *Fortune*, March 1977, pp. 31-82.
- Lecht, C. P., *The Waves of Change: A Techno-Economic Analysis of the Data Processing Industry*, New York: Advanced Computer Techniques Corp., 1977.
- Lynn, Frank., "An Investigation of the Rate of Development and Diffusion of Technology in Our Modern Industrial Society," *Report of the National Commission on Technology, Automation, and Economic Progress*, Appendix Volume II, Washington, D.C., 1966.
- MAPTEK *Strategy Report*. San Jose, CA: Quantum Sciences Corporation, December 1975.
- McCarter, P. M., "Where is the Industry Going," *Datamation* 23 (2), February 1977, pp. 63-65.
- Rosenthal, P. H., *A Cross-Impact Simulation of an Emerging Industry*, Unpublished dissertation, University of Southern California: Los Angeles, 1979.
- Shubik, M., *A Business Game for Teaching and Research*, New Haven, CT: Cowles Foundation, Yale University, 1964.
- Spencer, M. H. and Siegelman, L., *Managerial Economics*, Homewood, IL: Richard D. Irwin, Inc., 1964.
- Tintner, G., *Econometrics*, New York: John Wiley & Sons, Inc., 1952.
- Turn, R., *Computers in the 1980's*, New York: Columbia University Press, 1974.
- Wahi, P. N., "A General Management Business Simulation in APL," *IBM Systems Journal* 11 (2), 1972, pp. 169-180.
- Withington, F. G., "IBM's Future Large Computers," *Datamation* 24 (7), July 1978, pp. 115-120.



# Organization of the TRAC processor-memory subsystem\*

by R. N. KAPUR, U. V. PREMKUMAR and G. J. LIPOVSKI

University of Texas  
Austin, Texas

## INTRODUCTION

TRAC integrates a sufficient number of architecturally unique features to justify the development of microprogrammed processor and memory modules. This paper contains a discussion of the issues considered in the design and implementation of these modules.

In this section a list of design objectives and constraints that guided the system designers is given. The rest of the paper is divided into two major sections: functional specification and implementation. The functional specifications are derived from overall system design objectives and the constraints imposed by sibling subsystems. The implementation section focuses on the more important realization issues and presents specific solutions based upon these considerations.

A concluding section contains a status report on the hardware development and a short subsection on the testing methodology used while fabricating the modules. The appendix contains specification of registers in the memory and processor modules. In addition, it describes the microword format and explains some of the microinstructions.

### *Design objectives*

Varistruure and reconfigurability<sup>6</sup> are the cornerstones of the TRAC system. Reconfigurability has major repercussions on the switch design. Varistruurability affects processor design. Under varistruurability an  $n$ -byte operand can be processed by one or more byte-wide processors (Figure 1). The opcode that directs these operations must be independent of the physical structure of the machine. Therefore the processor must contain status information for guiding the microcode that realizes the opcode. Additionally, if the processing of an element is spread over a number of processors, control of data transfer between processors is necessary (e.g. carry propagation for addition as in byte slice processors).

TRAC provides support for SISD and SIMD<sup>3</sup> modes of operation. SIMD operation implies that all the constituent processors must be in lockstep from operation to operation. I/O and other operations whose time of execution is non-

determinate violate this condition by operating in a 'semi-synchronized' mode. Here the command is issued at the same instant to all the processors in an ensemble. Thereafter the processors go out of lockstep at the microcode level. When all the processors complete the operation, they must be synchronized to come back into lockstep.

The provision of hardware support for virtual memory management removes a major run time burden from operating systems. Paging strategies based on page usage counts can be realized using replicated hardware: parallel searching can then provide better performance than software table lookup and update techniques.

Lastly, the flexibility of a multiple processor system is closely related to the generality of the underlying communication subsystem [1]. However, this subsystem is traditionally the most expensive component of a multiprocessor system. Channels must therefore be developed that fit into the existing subsystem with minimal incremental cost and no degradation in performance of the original subsystem.

### *Design constraints*

In keeping with the philosophy of avoiding nonarchitectural pitfalls, this machine has been fabricated using standard SSI and MSI TTL components wherever feasible. The design, however, is LSI compatible using the following criteria:

1. The entire system is fabricated from four types of subsystems: processor, switch, primary memory and self managing secondary memory. (The last two subsystems are also referred to as MIO modules.)
2. Each subsystem is made as intelligent as possible within the restriction of the gates/chip criterion.
3. Pinout restrictions have been strictly adhered to: the processor and memory modules interface to the rest of the system through an 8 bit port per module. Less than 10 additional lines are needed per module for power, ground and control signals.

## FUNCTIONAL SPECIFICATION AND FUNCTION DISTRIBUTION

This section begins with a subsection specifying various functions that have to be realized by the processor-memory

\* This research is supported under NSF Grant MCS77-15968

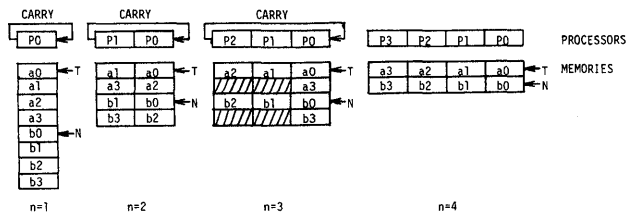


Figure 1—4 byte operand processing on  $n$ -byte wide machine.

subsystem. These functions include the basic functions provided in a conventional computer system as well as the unconventional features needed to implement a reconfigurable, varistructured architecture which supports virtual memory and interprocessor communications. The second subsection describes the distribution of these logical functions among different modules comprising the processor-memory subsystem.

### Functional specification

The processor-memory subsystem of TRAC realizes a multitude of logical functions to achieve a flexible and powerful multiprocessor system. These functions may broadly be divided into two classes: the conventional operations expected in a single sequential processor system and the additional operations required to support a multiple processor system.

A conventional system consists of an arithmetic and logic unit, a set of fast access registers, primary memory, secondary memory, input/output units, a collection of operational status information, datapaths, control and sequencing logic. Further, most modern systems provide interrupt handling, some virtual memory support and supervisory mode of operation.

In addition, the TRAC system provides the following functions to support variable structure and multiple modes of operation: processor linkage via instruction and shared memory trees, interprocessor communication through packet transmission and reception. These functions lead to additional operational and structural status information, datapaths, control and sequencing logic.

### Function distribution

The banyan network provides efficient interconnection between its apex and base nodes. By placing resource modules at both ends, the delays of propagation are reduced. Further, processing functions are different from memory and I/O operations. The number of distinct modules in a multiple processor system should be small. Thus, most of the functions have been divided between three modules, the Processor Module (PM) and the MIO modules. The names only indicate the primary functions and each module realizes several additional functions. A processor module may be connected to several MIO modules by a data tree on the switch.<sup>6</sup> Such a connection facilitates the execution of a 1 byte-wide

SISD task by providing a dedicated data and control linkage between the processor and the MIO modules. Several such connections may be used to support the execution of multi-byte SISD and SIMD tasks by connecting the processor modules via an instruction tree.

A 1 byte-wide SISD task environment consists of a single processor and several MIO modules. The processor controls and accesses the primary memory and the self managing secondary memory via the connection provided by the data tree. The functions associated with the primary memory module are first explained. The TRAC address space is divided into four separate functional spaces: OS, DS, CS, and PS. Associated with each space are two pointer registers for memory access within that space. A primary memory module has 4K bytes of primary memory divided into four physical pages of 1K each. Each physical page may be assigned to any logical page in any functional space. Each primary memory module maintains a copy of the eight pointer registers and these are consistently updated. Additional hardware is provided to evaluate and retain page usage to aid in page replacement policy implementation. The architecture provides a flexible and efficient distributed memory system consisting of functionally independent modules with minimal interactions. The self managing secondary memory (SMSM) module provides virtual memory backup and segmented file I/O. It is based on the concepts described in [2]—further description of this module is beyond the scope of this paper.

Each processor module can operate either independently or as part of a set of processors. It has a byte sliced Register-Arithmetic-Logic-Unit (RALU) and associated operational status; for example, the condition codes and interrupt status. The processor uses microprogram control and sequencing. The basic clock and associated phases are derived from the master system clock.<sup>9</sup>

Additional functions provided in the processor and MIO modules to support multiple modes of operation are discussed next. The instructions tree provides a broadcast bus and a GPC tree among a set of processors. Though the primary function of the broadcast bus is opcode transmission, it is used in other data transfers also. The GPC tree provides a flexible mechanism for processor linkage: it is used for selective grouping of the processors into subsets (each subset processes a different element), collection and distribution of status, processor synchronization and prioritization for shared memory access; the original function of GPC trees of linking processor carries for multibyte operations is retained. Variable structure is implemented by partitioning the set of processors into groups and breaking linkage at appropriate processors. Relevant structural status is stored in each processor for an efficient realization of this scheme. A set of processors may share an MIO module via a shared memory tree. An MIO module may be identified as a shared module and includes status information to specify ownership. A processor may be connected to more than one shared MIO, and a 'coloring' scheme is used to avoid deadlocks in shared module access. Thus a programmer can distinguish between several shared trees by assigning different 'colors' to them.<sup>9</sup>

Interprocessor communication<sup>8</sup> is another major function provided in the TRAC system for a flexible and comprehensive interaction among different processors. Each processor is given a physical address and any processor can send a packet to any other processor. The sending processor instructs a specified MIO module within its data tree to send a packet and provides the necessary parameters. The packet transmission logic and dispatch buffers are associated with the MIO module, whereas the packet reception logic and reception buffers are associated with the processor. There are two separate channels for two types of communications. These channels are called the mapping and interrupting channels with respect to their primary use.

**IMPLEMENTATION ISSUES**

We can now describe the primary memory module and the processor module implementation details. The description will be done at a block diagram level; specific IC details have only been included where they have materially influenced the design.

*The primary memory module*

The primary memory module decodes and executes micro orders that a processor module broadcasts on a data tree (Figure 2). The micro order is issued in switch phase 0s2 of a system cycle; the data from the resultant operation is transmitted in 0s3 and 0s4 of the next cycle (Figure 3). Associative addressing within modules determines exactly which module performs the operation. The next micro order from the processor is transmitted before the data transmission—this overlap pipeline is used to mask delays that are introduced by the switch.

Primary memory micro orders can be broadly divided into two classes:

1. Pointer memory instructions—these instructions are used in accessing memory and modifying pointers (e.g. IRT: pre-auto-increment read using T pointer).

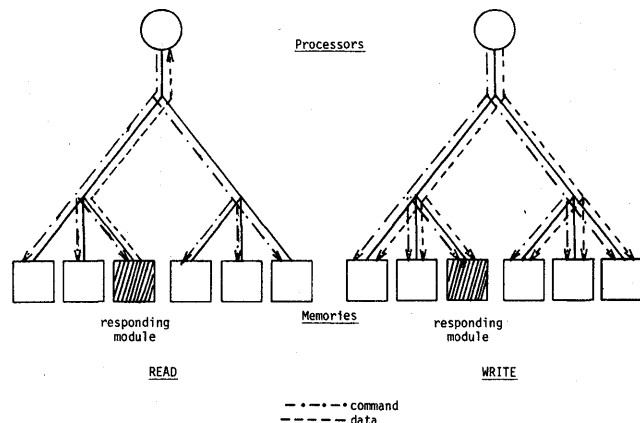


Figure 2—Processor memory operation.

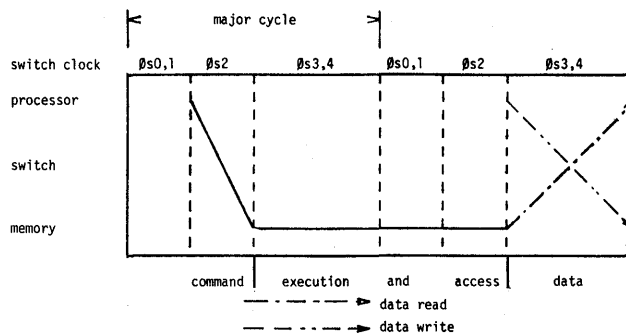


Figure 3—Processor memory operation timing.

2. Status instructions—these instructions are used for loading and storing status information (e.g. WPST: write page status using T).

Each TRAC primary memory module contains four pages; a page contains 1K (1024) 8 bit bytes. Each page has an associated space type-page number register (SPNR-8b), page usage counter (PUC-4b), and a page status byte (PSB-4b). An 8 × 16b pointer register file in each module stores pointers used for addressing. Additional registers associated with a module are module color (MCR-3b), module status (MSW-5b), and task status word (TSW-3b). A packet buffer (PB0 . . . PB3-4 × 8b) is provided in each module for assembling packets.

The appendix contains a detailed description of the primary memory registers and their functions.

Once a micro order is received by the modules in a data tree the sequence of operations in each module is as follows: the controller extracts the contents of an appropriate pointer register into the arithmetic unit where the address is modified as required. This address is transferred to the associative addressing section that contains four page number registers and four comparators. If no match occurs in any of the modules, the processor module is interrupted via a separate line that follows the data tree. If a match does occur in any one of the modules it transmits a 'no interrupt' to the processor module; this signal overrides the 'interrupt' signal from the other modules. The selected module now sends/receives the relevant data.

The timing of this operation sequence is as follows: the address manipulations are performed during 0s3 and 0s4 after the command is received. The effective address is generated just before 0s0. After an access delay that can last through 0s0, 0s1 and 0s2, data is available to be placed on the switch at the beginning of 0s3. It is transmitted through the switch during 0s3 and 0s4 and arrives at its destination before the end of 0s4 (Figure 3).

Two additional 'nonmemory' operations can be performed by a primary memory module:

1. It can be used as the source for interprocessor packets: a packet assembly buffer and packet generator bits are provided for this.
2. When it is used as the root of a shared tree it performs

functions pertaining to shared tree colors—these operations are of relevance in providing deadlock free access to a shared resource.

Figure 4 illustrates the structure of a primary memory module. The block labelled 'switch interface' is a scaled down version of a switch node [9].

#### The processor module

This section deals with the block level organization of the TRAC processor module. As described earlier, this module realizes both the conventional and unconventional features of the TRAC system associated with the processor. Figure 4 shows a schematic diagram of the block level description. The module is essentially made up of a RALU, status, packet buffer, control store, sequencer and several control units.

The TRAC architecture allows a powerful and flexible instruction set and the implementation is realized mostly through LSI and MSI components. Thus, microprogram control is best suited for its design. Here the microprogrammer's view of the system is presented and no effort is made to explain the hardware details. Owing to the limitation of space, only the essential features of the microarchitecture are presented.

The RALU contains two 2901 bit-slice processors linked together through carry look ahead logic. The registers of the 2901 are used by the microprogram and are generally invisible to the machine language program. This is necessary to allow a virtual machine that can accommodate variable word widths requiring different 'heights.' Six of the 17 internal registers are used as microprogram accumulator and working registers. Four of the registers are used as counters and the

remaining to hold constants. The counters are used to govern multiprecision, vector and loop operations. The constants are usually structural parameters specifying the variable structure and may be loaded under user control. The microcode permits the programming to load and store these registers from the data bus, do arithmetic and logical operations and move data among the registers. The microcode is designed to realize all the RALU functions needed to efficiently implement the TRAC instructions set. the RALU operation may be conditionally executed under microcode control, i.e. it may be suppressed or enabled depending on some status bits. This enables selective masking of processing in specific processors and is useful in multiprecision and vector operations. It is possible to choose one out of several alternative RALU operations by substituting some bits from the instruction opcode into the micro order. Thus it is possible to implement several machine instructions essentially with the same microcode.

The TRAC system is a memory to memory architecture. The processor accesses operands (or fragments of operands), operates on them and replaces the result in the memory. Though the memory may be distributed, its location is transparent to the processor and appears as a centralized unit. The processor sends a memory command specifying a service: for example, read from or write in a specific location in a space, pointed by an address register. There exist inherent delay in the network as well as in the memory, and the request is serviced a cycle after the request. This results in a one step pipeline between the processor and the memory. The microcode should take this one cycle delay into account in the implementation of microalgorithms.

The complexity of TRAC architecture, specifically the variable structure and multiple mode of operation, lead to substantial status information that needs to be kept in the processor for efficient execution of microalgorithms. The status information may be broadly divided into two classes: operational status and structural status. The operational status is dynamic and includes element condition codes, vector condition codes and processor condition codes and interrupt status bits. The vector condition codes and element condition codes are programmer visible and are used by branching and masking instructions. The processor condition codes are the dynamic status bits used by the microprogrammer in conditional execution of RALU operations, accumulation of temporary Boolean variables and conditional sequencing of microinstructions. The interrupt status bits are used to interrupt the microprogram on page faults, packet arrivals or I/O interrupts. The structural status is relatively static and is indirectly influenced by the instructions that specify the structure of the system. For example, the processors carry flags to indicate where the carry linkage is to be broken in the chain of processors within a task. There are status bits to determine which processors operate on least/most significant bytes of an operand (or an operand fragment). Further, a specific processor may be identified as a task head and be given special responsibilities. Other status bits are used to mask specific processor operations in specific cycles under microprogram control.

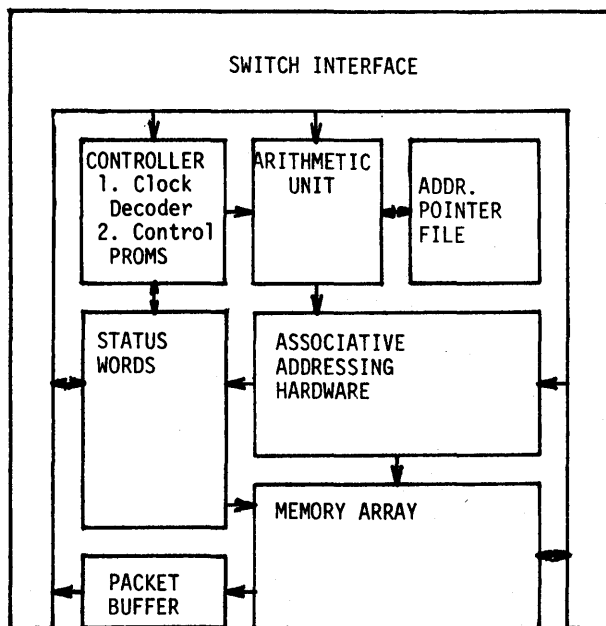


Figure 4—Structure of the memory module.

The GPC tree is described in [5]. A set of processors can use the GPC tree connection to realize multiprecision and SIMD operations. In multiprecision operations, the GPC tree is used as a carry lookahead linkage and this permits reasonably fast arithmetic and shift operations. The structural status is used to break the carry linkage at appropriate processors within the ordered set. In SIMD operation, the GPC tree is used to collect and distribute status information among processors. For example, a branch condition may be collected in the task head and transmitted to all the processors, so that all the processors choose the same microword in the succeeding cycle. Further, the GPC tree is used to resynchronize a set of processors after the completion of an asynchronous operation as in mapping packet communication [8]. The GPC tree is also used to resolve contention and permit mutually exclusive access to shared memory [8].

The packet reception unit is essentially an independent unit and consists of packet buffers to store incoming packets and generate a prioritized packet arrival interrupt. The packet buffers may be addressed and loaded in the memory under microprogram control. Separate buffers are present for mapping and interrupting packets. The mapping channel is used primarily for data redistribution within the domains of several processors. The interrupting channel is used for signaling and messages. The bus control unit controls the communication on the data bus by specifying at most one source and possibly many destinations.

The micro sequencer consists of two 2911 sequencers and associated PROMs and selectors. A system cycle consists of a single memory cycle and two processor cycles. These processor cycles are called phases A and B. The microwords for phases A and B have successive even/odd addresses.

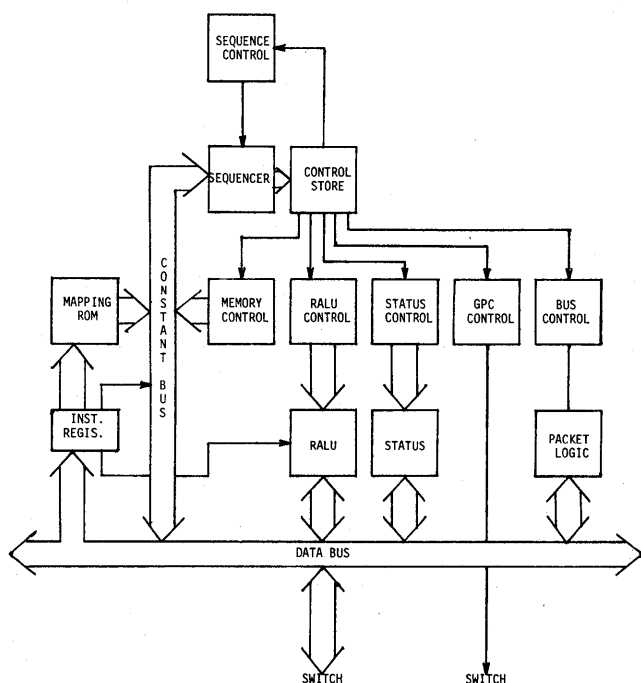


Figure 5—Structure of the processor module.

Thus the micro sequencer provides the most significant eight bits of the microword address and the least significant bit is derived from the phase status. This provides up to 512 microwords for the control store. It is possible to have specific micro sequences that implement interrupt service etc., that can be stored in the higher address space in a 1K long control store. The microword is four bytes long and its specific format is given in the appendix. The sequencer is provided with several useful sequence control commands. These include, continuing to the next micro instruction, jumping conditionally or unconditionally to a specific address, setting loops, calls to subroutines and returning from them. Thus the micro sequencer provides a full repertoire of useful control structures.

The processor controls the SMSM in a manner similar to that of primary memory. Some of the MIO modules are equipped with SMSM subsystems and accept secondary memory commands from the processor and respond with service. The interaction between processor and SMSM is asynchronous and uses handshaking. During I/O operations, the processor performs the DMA function and is dedicated to that process. The processor accepts interrupts from the SMSM and has means for reading its status through microprogram control.

## CONCLUSION

The processor and memory modules have been fabricated with standard SSI, MSI and some LSI parts. A microcomputer system with dual floppy discs was used as the heart of the development and testing system.

We have developed a microcode assembler from the very versatile macroassembler on our system. A 144 bit wide I/O port was fabricated to interface the subsystem under test to the computer. Hardware was exercised by software modules that emulated those sections of hardware that were not yet ready. The software/hardware boundary was moved back as additional hardware was developed until the entire subsystem was complete.

## REFERENCES

1. Agerwala, T. K. M., and Lint, B. J., "An overview of communication issues in the design and analysis of parallel algorithms," (submitted for publication).
2. DeMartinis, M. et al., "A Self Managing Secondary Memory," *Proc 2nd Ann Symp on Comput Archi*, 1973.
3. Flynn, M. J., "Very high speed computing systems," *Proc. of the IEEE*, 54, pp. 1901-9.
4. Lipovski, G. J., "A varistructured array of microprocessors," *IEEE Trans on Comput.*, vol c-26/2, Feb. 1977.
5. —, "Optical Linkages for Interconnecting Integrated Circuits," *Proc. of the NCC*, 1977.
6. Lipovski, G. J., and Tripathi, A. V. R., "A varistructured reconfigurable array computer," *Proc Parallel Processing Conf.*, 1977.
7. Sejnowski, M. C., et al., "An overview of TRAC," *Proc. of the NCC*, Vol. 49, 1980.
8. Premkumar, U. V., et al., "Interprocessor communication in TRAC," *Proc. 1st Intl Conf on Distributed Comput Systems*, 1979.
9. Premkumar, U. V. et al., "Design and Implementation of the Banyan Interconnection Network in TRAC," *Proc. of the NCC*, Vol. 49, 1980.



## APPENDIX

*Primary memory registers*

The page registers are organized as follows:

1. SPNR [0. .7]—space type page number register
  1. 0. .5: page number of 1K page
  2. 6. .7: space type
    1. 00 Program space
    2. 01 Control space
    3. 10 Data space
    4. 11 Operand space
2. PUSW [0. .7]: Page usage/status word.
  1. [0. .3]: Usage count.
 

This counter records the reference frequency for a page. It is zeroed when a page is loaded and is incremented on every  $n$  accesses to that page. This counter saturates at 1111 binary.
  2. [4. .7]: Status
    1. 4: Read Write/Read only
    2. 5: Supervisor User/Supervisor only
    3. 6: Clean/Dirty: indicates if a page has been written into
    4. 7: Locked/Free: indicates whether a page is tied into primary memory for the duration of the task.

The module register organization is as follows:

1. Pointer register file
 

This file contains the eight pointers used for memory addressing: T,N,X,Y,D,S,P, and W.
2. MCSW [0. .7]—Module color and status word
  1. [0. .2]: color
 

These bits are used in shared memory acquisition.
  2. [3. .7]: Status
 

These bits are module related and are different from module to module.

    1. 0: Shared/Local module
    2. 1: Owned/Unowned module (valid for shared module only)
    3. 2: APG/APG'—identifies asynchronous packet generating module
    4. 3: SPG/SPG'—identifies synchronous packet generating module
    5. 7: MPI: Module pointers invalid—this bit indicates whether the pointer registers in a module are valid. If this bit is true only loads to registers are permitted for that module. Once all registers are loaded this bit is automatically made false.
3. TSW [0. .2]—task status word.
 

These bits are task related and are identical in all modules in a task.

  1. 0: Supervisor/User mode
  2. 1: W/W\* mode: in the W mode, W points into PS. in the W\* mode, W points into OS.
  3. 2: Primary/secondary access: this bit is for performing primary memory operations

*Processor registers**RALU registers*

1. A0—Accumulator
2. A1—Auxiliary Register
3. A2-A5—Working Registers
4. SC—Slice Count
5. SCI—Slice Count Initial value
6. BCH—Band Count High byte
7. BCL—Band Count Low byte
8. BIH—Band count Initial value High byte
9. BIL—Band count Initial value Low byte
10. LC—Loop Count
11. N—Number of processors in a group
12. PN—Processor Number
13. GN—Group Number
14. BN—Byte Number

*Status registers*

1. ECC—Element Condition Codes
2. VCC—Vector Condition Codes
3. PSW0—Processor Status Word 0
4. PSW1—Processor Status Word 1
5. PSW2—Processor Status Word 2

*Packet registers*

1. SPR0-SPR3—Synchronous Packet Registers
2. APR0-ARP11—Asynchronous Packet Registers

*TRAC microinstruction specification*

The TRAC system is being designed to operate at a 1 MHz system clock, i.e. a 1 microsec system cycle. A system cycle consists of two processor cycles, phases A and B. The microinstruction formats are somewhat different for the two phases and are as given below in Figure 6.

TRAC control store consists of 1K microwords of 4 bytes each. The TRAC instruction set utilizes the low order 512 words of the control store. The high order 512 words are reserved for special microsequences used in interrupt service etc. The phase A and phase B microwords occur in successive even-odd locations. Thus the sequencer provides only

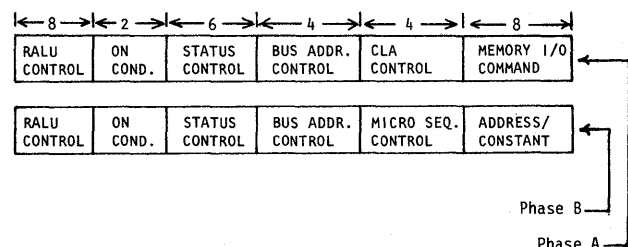


Figure 6—Microword format.

the high order eight bits of the next microword address. The least significant bit is derived from the current phase. Thus the control transfers can be made only to the even numbered address locations.

The first field specifies the RALU operation, the second field specifies a condition on which the RALU operation is enabled. In phase A, the third field contains a command to the primary memory, the SMSM or both. In phase B, the same field contains a constant which may be used as a jump address. The fourth field consists of the status command that selects which status is to be updated. The fifth field is for bus control and determines which unit drives the data bus. The sixth field is interpreted as the control command to the

GPC link in phase A and as the sequencer command in phase B.

There are two busses in the processor, the data bus and the constant bus, both of which are eight bits wide. The data bus may be driven by RALU, the switch, the status registers, the packet registers or the constant bus. The data bus is accessible to the RALU, the switch, the status registers, the packet registers and the instruction register. The constant bus is driven by the control store or the mapping ROM and it is possible to substitute the least significant three bits from the instruction register. The constant bus is accessible to the data bus and the microsequencer. It is also used as an extended command code in certain operations.



# An overview of the Texas Reconfigurable Array Computer\*

by MATTHEW C. SEJNOWSKI, EDWIN T. UPCHURCH, RAJAN N. KAPUR,  
DANIEL P. S. CHARLU and G. JACK LIPOVSKI

*University of Texas at Austin*  
Austin, Texas

## 1.0 OVERVIEW

This paper presents an overview of TRAC and then discusses the system's suitability for some promising applications: Monte Carlo techniques, numerical solutions to linear systems, and data base applications.

### 1.1 Introduction

The Texas Reconfigurable Array Computer (TRAC) is an experimental computer system currently being built at the University of Texas at Austin. It is a multiprocessor system intended to utilize existing microprocessor technology and yet provide performance that promises to be superior to current state-of-the-art processor capabilities. The uniqueness and the potential capabilities of TRAC arise from its interconnection network; a dynamically reconfigurable banyan network.<sup>1</sup> The banyan network serves to partition and to configure the processor, memory and I/O resources of the system into different architectural organizations as demanded for efficient application formulation and solution. Although it was originally formulated as a high capacity scientific computer, it has been shown to perform well in both numerical and non-numerical applications.

A high-performance microprocessor-based system has some definite advantages over other large-scale computer designs. Chip count tends to be low, and microprocessors are very competitive in the "cost per computing power" sense, and should continue to be increasingly cost-effective in the future. Therefore, an obvious possibility for high-performance design is to include in the system as many microprocessors as needed to achieve the performance level desired. A major stumbling block in such a multi-processor system has been the nature of the interconnection network. In fact, the interconnection network may be much more expensive and complex than the rest of system.

It is well-known how a program can be run on a set of processors such that the data operated on consist of multiple-precision vector elements with the entire vector accessed simultaneously. This is done by connecting  $n$  processors to each of  $n$  independent memories with common address

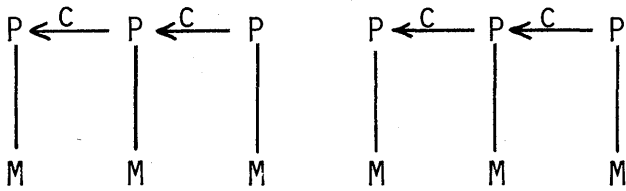
space, so that a given address across the memories contains one vector. Each CPU then processes one piece of the vector in parallel with all the others. Carry logic is used to interconnect processors acting upon different fragments of the same vector element, to achieve multi-precision arithmetic. For example, Figure 1 shows a system that operates on 2-element vectors where each element is 3 bytes long (each processor handles one byte).

The major drawback to such a machine is its lack of flexibility. The machine is essentially "hardwired" to operate on a specific type of data. When other data types are operated upon, gross inefficiencies result; the machine either ends up underpowered, overpowered, or just poorly configured for the task (for example, using the machine in Figure 1 to process 7-element vectors of 8 bits per element is clearly inefficient).

A logical way to improve the situation is to introduce switches into the bus structure, under control of software. One general way of doing this is the crossbar arrangement as shown in Figure 2.<sup>2</sup> Now the machine can be dynamically configured to adapt itself to the needs of the task. The big disadvantage here is that the number of switches required is proportional to the square of the number of resources in the system. For large systems (for example, a 1000-processor system), the switch count is then prohibitively large.

The TRAC system being built is a form of switched network in which the number of switches required goes up as  $n * \log n$  with respect to the number of resources to connect. The major unique features of the TRAC computer are space sharing, reconfigurability, varistructuring, inter-task communication ability and the fact that its design makes it a virtual machine to the user. Space sharing implies that independent or interacting tasks can all be running simultaneously on the same computer, as opposed to the time sharing where tasks must await their allotted time-slot to execute. Reconfigurability is the ability of TRAC to dynamically partition its processors and resources (primary and secondary memories, I/O devices) under software control to obtain optimal utilization and minimal waste for the set of tasks to be run. Within a partition, TRAC is varistructured in that regardless of the data structure requirements for the task, any data width or architecture may be used; flexible microcode makes the exact processor configuration transparent to the software for the task. The machine is virtual

\* This research is supported under NSF Grant MCS77-15968.



P = processors, 8-bit

M = memories

c = carry linkages

Figure 1—A two-element 24-bit/element vector computer.

in that user programs can be oblivious of the specific set of memory and processor modules used. Memories have space-page registers which allow them to be combined in any way to form address spaces. An extension of the virtual nature of the machine is TRAC's modularity. Without changing operating system software, resources and processors can be added on to the system in a building-block fashion to expand the machine to meet the needs of the user.

We now review the concepts which TRAC uses to achieve these goals. It is important to realize that these concepts are each more general in potential than in their realization in TRAC. They are currently the topics of extensive research by TRAC personnel at the University of Texas.

1.2 Concepts

The most fundamental concept in the system is the use of an SW-banyan network<sup>3</sup> to interconnect the set of processors

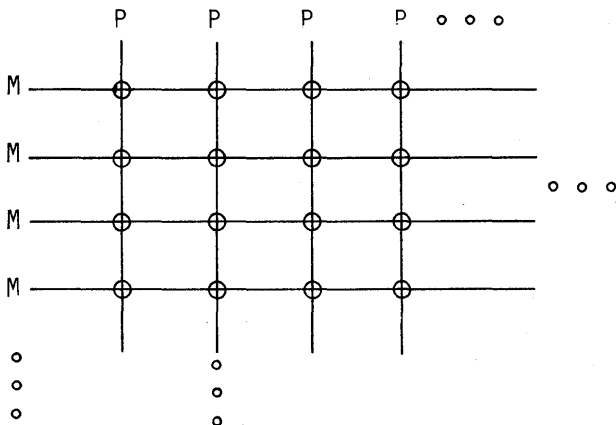
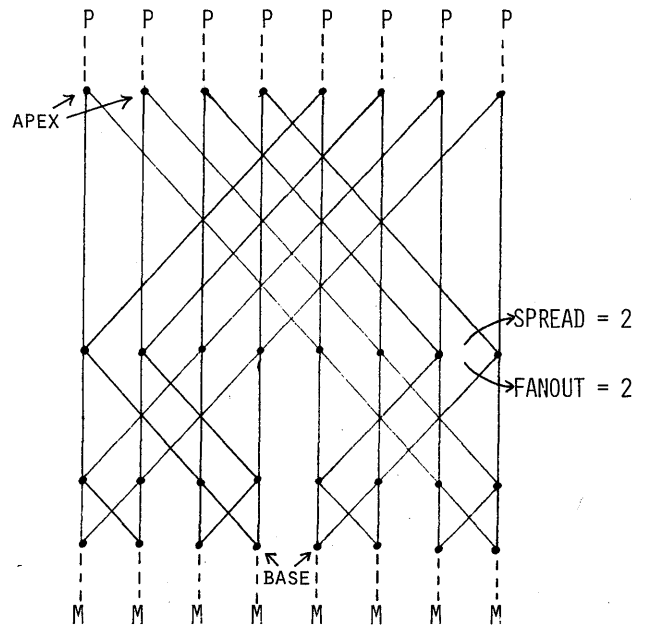


Figure 2—Crossbar network.

with the set of resources. A banyan is represented by a graph in which nodes are divided into three types: apex, base and intermediate nodes. In TRAC, apex nodes represent processors, base nodes represent memory or I/O or combined memory-I/O resources and other nodes represent switch nodes. It has the property that there is a unique path between any apex (processor) and base (resource) node pair through the intermediate nodes (switches). Furthermore, the graph may be regular in that all apex nodes look alike, all intermediate nodes look alike, and all base nodes are either memory nodes or I/O nodes or combined memory I/O nodes. Note the potential for mass production of the components; in fact, only four types of LSI IC's would be needed to build a very large and powerful computer. The intermediate nodes have a fixed number of arcs going toward apex nodes ("spread") and going toward base nodes ("fanout"). Figure 3 shows a typical SW banyan network. A characteristic of SW-banyans is that each level of intermediate nodes has  $O(n)$  nodes, and there are  $O(\log n)$  levels, hence  $O(n \cdot \log n)$  switches.

Each switch in the network is capable of interconnecting an incoming arc to any subset of the outgoing arcs and vice versa. Initializing the switches is done in hardware and is transparent to user programs.

For a given task, three types of subtrees can be established using the switch nodes': data trees, instruction trees, and



- P - Processors
- M - Memories & I/O Devices
- - Switches

Figure 3—A typical SW-banyan.

shared memory trees. Physically these are actually trees, but once formed they logically and electrically perform as buses. The data tree is used as a data bus to connect a processor with appropriate memory modules. The instruction tree is used as an instruction bus to broadcast instructions to a set of processors performing the same task (SIMD mode). Shared memory trees connect a set of processors to a single memory module for the purpose of sharing data; parts of the shared memory tree are used to extend a data bus from time to time in order to share a memory. The operation of these subtrees is shown later. Figure 4 shows examples of these trees. Following the paths of instruction trees and shared-memory trees is a GPC (generate-propagate-carry) bus used to interconnect processors all working on the same multi-precision data element. The GPC bus is also used for synchronization of processors, distributing status information and arbitration over control of shared memories.

In addition to the primary memories, TRAC has self-managing secondary memory (SMSM).<sup>4</sup> These modules are in-

cluded in the system network as resources. They are low cost, medium speed memory devices which may initially be implemented using CCD's, bubble memory or MOS RAM technology. The data within the SMSM is organized as segments and accessed by means of segment labels, making it act like an associative memory over the labels only. Since the directory structure is implemented in hardware, the burden of locating specific sets of data is taken from the software. A key use of SMSM's in the TRAC architecture is to virtualize the memory. If a processor accesses memory data which is not currently in one of the memory modules attached to the processor, the SMSM is used to page out current data in one of the modules, and then read in the desired page of data into the now-free memory module.

The concept of index registers located in memory nodes is used in TRAC. Instead of having processors supply a complete address and data bus to all resources (which would have to be replicated on each arc of the SW-banyan), a smaller 8-bit bus is used. References to locations in memory modules are made by specifying one of the index registers which reside on the memory modules themselves. This technique allows a short (8-bit) macro-instruction to be sent through the switches rather than a longer (16-bit) address. This includes the program counter and data stack pointers, as well as other special-purpose registers. The modules also have built-in hardware to increment or decrement the registers. Thus, in straight-line code, there is never any need, for example, for a processor to take part in the incrementing of the program counter (other than instructing the memory module to do so). Stack operations, also, typically require only incrementing and decrementing operations on the stack pointers.

Now that we have touched upon the concepts in use by TRAC, we describe the prototype implementation now in progress at the University of Texas at Austin and the configuration of architectures for tasks and task execution.

### 1.3 Operation

The TRAC system will initially consist of a four-level SW-banyan network with a fanout of 3 and spread of 2 (this amounts to 211 switch nodes). The network will connect 16 processors to 81 memory-I/O nodes. The base nodes will all (except for the switch control node and the port to external computer systems) have uniform interfaces which may support a memory module (of 4K byte capacity), an SMSM of 64K bytes or both. The banyan arcs consist of 23 signals, 8 of which are the data bus and three the carry-lookahead logic.

Given a set of tasks to run on TRAC, sets of resources can be allocated to each of the tasks in such a way that the partitions all are independent of one another. Thus the tasks "space share" the machine; see Figure 5. Each task has exclusive control of its resources and does not have to contend with other machines for the use of these resources. In addition, while the tasks are running, the system is dynamically reconfigurable, allowing specialized task structures to be created on demand.

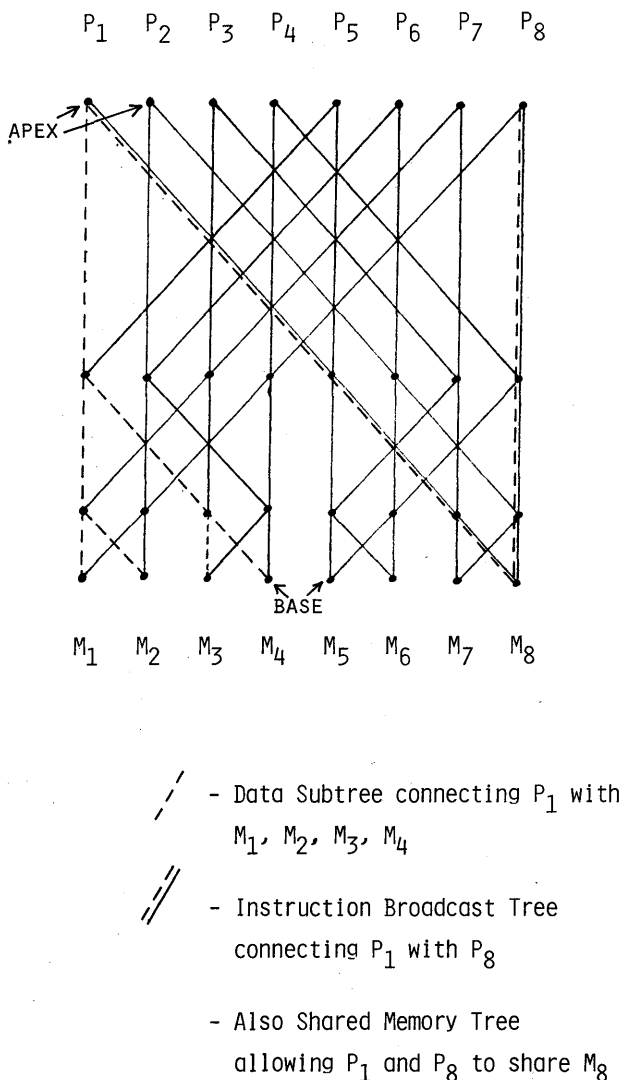


Figure 4—Subtree examples.

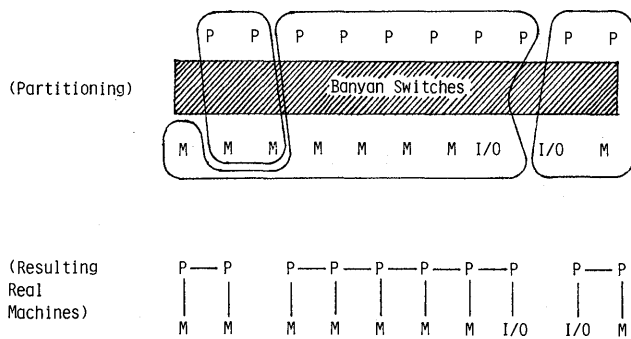


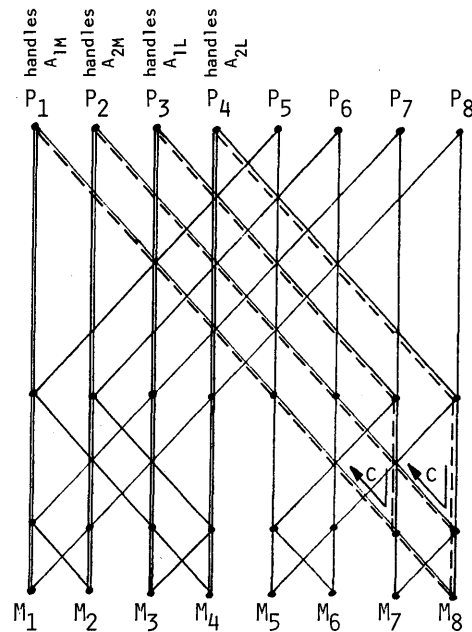
Figure 5—Partitioning and space sharing of resources.

Each processor in the TRAC system operates with 8-bit operands. To achieve parallelism in processing multi-precision data, TRAC uses multiple processors. First we consider processing vectors of  $v$  elements of  $p$  bytes precision per element, using  $v \cdot p$  processors (later we show how less processors can be used by “folding” the array and/or its elements). Since all the processors will perform nearly the same operation on their respective bytes of a vector simultaneously, an instruction tree connects all processors together during an instruction-fetch cycle. The task’s code thus needs to exist in only one of the processor’s memory spaces. That memory fetches an instruction, then broadcasts the instruction through the data bus and over the instruction bus in one memory cycle to all the other processors in the partition. Figure 6 shows an example of a subcomputer that processes 2-element vectors, 16 bits per element. Note that the instruction subtree is totally independent of the data subtree. The GPC bus included in the instruction subtree provides carry-lookahead capability as well as other synchronization functions as described above.

Thus the entire array can be accessed simultaneously. In many cases, the number of available processors is limited, or the size of the vector is too large to allow completely parallel processing of the elements. This is where TRAC’s varistructured capability comes into play.<sup>5</sup> When a task is to be loaded and run, it passes information to the scheduler for TRAC concerning the type of data structure the task uses as well as the urgency of the task (the more urgent a task is, the more processors it is allocated). The scheduler arbitrates between all the tasks and performs a partitioning of processors and resources in order to best satisfy all the tasks’ needs.

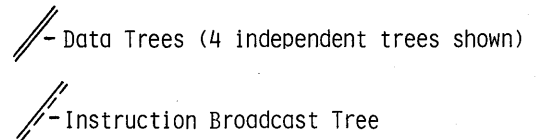
When a task is allocated a number of processors fewer than the full width of its vector elements, the elements are “folded” into the available memory width. For example, Figure 7 shows to different ways in which a 4-element vector with 2-byte elements can be packed into the memory modules of 6 processors (naturally, if 8 processors are available, the packing would be much more straightforward). The microcode in the processors makes the packing algorithm transparent to the user; for example, adding two vectors together would be accomplished by the same machine-language instruction regardless of which of the two alternative packings shown in Figure 7 is used.

More generally, logical vectors can be packed into a fixed



$$\vec{A} = (A_1, A_2) = (A_{1M}, A_{1L}, A_{2M}, A_{2L})$$

(16-bit values) (all 8-bit values)



Carry Flow shown by arrows

Figure 6—A subcomputer that handles 2-element vectors, 2 bytes per vector.

memory width (as specified by the number of processors allocated for the task) as shown in Figure 8. In the figure, each horizontal row of data is capable of being accessed simultaneously by the microcode; a reference to a vector at the machine language level translates to sequentially accessing each of the rows of the memory. The processors

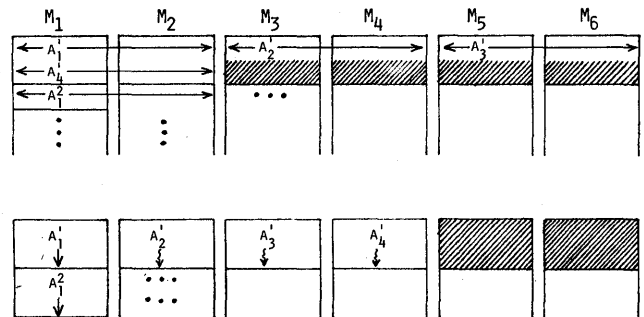


Figure 7—Two possible memory arrangements for 4-element vectors, 2-byte elements, in 6 memory modules.

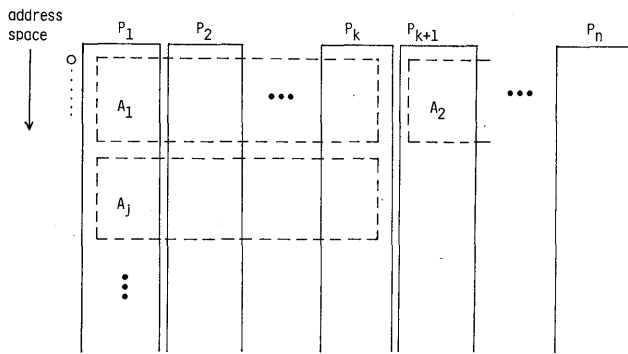


Figure 8—Varistructured processing of multi-byte precision vectors.

once again perform identical operations; furthermore, they repeat the identical operations at each row of memory until the entire segment is accessed. The only differences are when handling most-significant or least-significant bytes; the processor microcodes handle these automatically.

A data tree always has a processor at its root and one or more of the resources at its leaves. It is important to realize that though the interconnections are electrically tree-structured, it is used as a conventional bus through which the memories at the leaves of the tree are logically all mapped into the root processor's memory space which is always one byte wide, and potentially 64K bytes long. Each of the memory modules is 4K bytes long, and is divided into four 1K pages, each of which has an onboard page register telling where the page fits into the processor's memory space. The four pages may be dedicated separately as any of program space (instruction space), control space (which contains a stack for descriptor and control information), operand space (containing a task data stack) data space (containing task global data).<sup>6</sup> Each of the four data spaces have associated with them two pointer registers as previously discussed.

An instruction tree is, orientation-wise, upside-down with respect to the data tree. Its root is a dummy (i.e. unused) memory module, while all its leaves are processor nodes. Anything sent over this tree by one of the processors is broadcast to all attached processors in order to implement SIMD mode processing. The carry tree is the same structurally as the instruction tree. However, as discussed earlier, by forcing the generate and propagate to zero, the propagation of carry signals can be broken up allowing vector parallelism.

A third type of subtree, the shared memory tree, allows multiple processors to be simultaneously connected to a memory module. A shared memory tree is identical in form to an instruction tree. Attached processors may all be part of the same task or may be from different tasks. At any particular point in time, only one processor can have access to the memory. Arbitration for access rights to the memory are done round-robin style using the carry lookahead as a priority circuit, so that no processor can lock out other processors from access to the memory indefinitely. Shared memory provides a good means of communicating large amounts of intertask or intratask data.

Another means of processor-processor communication is packet switching.<sup>7</sup> Packet segments travel from memory node to processor node by hopping between adjoining switch nodes, one hop per memory cycle, independently of any trees that might be set up. The packet transmissions occur as a background activity so that they do not interfere with other activity. Destination information contained in the packets route the packet through the nodes. Packets may be explicitly sent and received through TRAC machine language or implicitly sent as part of certain microcoded instructions.

Since registers residing within the processors are of fixed width, but data can in general be of any width, TRAC machine language instructions that perform data manipulation do not explicitly involve processor registers (such as load and store accumulator instructions found on many other machines). Instead, the instructions tend to be the memory-memory type. Stack operations are supported, with the on-memory index registers used as stack pointers. For example, the add instruction takes the top two elements on the stack and replaces them with their vector sum, adjusting stack pointers accordingly.

TRAC subsystems can be architected to implement each of the types of parallel execution structures described in the literature.<sup>8</sup> In one type of asynchronous parallelism (MIMD), a task may, at some point in its execution, fork into multiple subtasks operating independently of one another. Once the fork occurs, subtasks may have to communicate data and status information with other subtasks. Another occurrence of asynchronous parallelism is the concept of task pipelining. In this scheme, a subtask operates on a set of input data and produces a set of output data. This output is then used as input to the next subtask in the chain, which likewise produces output. All processes operate asynchronously except for the necessary synchronization of inputs and outputs. A third form of asynchronous parallelism is the data-flow concept; extending the concept of operators to subroutines and functions (which are subtasks), a subtask is allowed to run only when all of its input parameters are present (the parameters may be outputs from other subtasks).

Asynchronous parallelism is easily achieved in the TRAC system. Because of the ability of subtasks to have completely separate and independent sets of processors and resources (space sharing), asynchronism and non-interference is assured. Several methods of inter-subtask communication are available, including shared memory modules, packet switching, and dynamic reconfiguration to physically transfer memory modules between subtasks.

Another mode of parallelism is vector parallelism, as discussed earlier. We have seen that the varistructured nature of TRAC allows it to intrinsically support this mode.

Synchronous parallelism includes situations in which subtasks may have different instruction flows, but must all operate such that simultaneous events are well-defined. For example, the addition of floating-point values might employ two subtasks; one to handle the mantissa and one to handle the exponent. If the mantissa subtask shifts the mantissa, the other subtask must simultaneously increment the exponent. Other examples include matrix inversions and string



or set searches. In the TRAC system, tasks running on different processors can be in synchronization with each other if care is taken by the system programmer to control task startup and interrupts. Inter-subtask communication could be implemented through shared memories.

The TRAC interconnection network also implements parallelism in data movement as well as computation. The many paths from apex and base nodes represent potentially active data paths for memory and instruction fetches. The use of the network for packet movement in its phases of inactivity for bus service offers a potentially vast bandwidth for asynchronous data movement.

## 2.0 APPLICATIONS

We now examine some typical application areas which could take advantage of TRAC's architecture. Each of these applications demonstrate the power of TRAC to support several forms of parallel execution, to be configured for a given application and to reconfigure during the phases of execution of an application and to utilize high-volume data transfers through the network. A crucial role for TRAC will be to serve as a focus for the analysis of applications and algorithms for parallel formulations. The existence of a system which can serve as an experimental facility for execution of parallel algorithms should be a strong stimulus for this significant problem area.

### 2.1 Iterative solutions of linear systems

We will describe the configuration of systems for the solution of large sparse linear systems of the type

- (1)  $Au = b$   
 where  
 $A$  is an  $N \times N$  coefficient matrix,  
 $u$  is  $N$  element vector,  
 $b$  is a known  $N$  element vector and  
 $N$  is of the order of  $10^3$  to  $10^6$ .

Such systems are frequently encountered in the numerical solution of partial differential equations arising from problems in areas such as oil recovery, nuclear reactor design, scientific weather prediction and many others. Partial differential equations (PDEs) can be converted into simultaneous linear equations using finite element methods or finite difference methods.<sup>9</sup> The conversion method, as well as the nature of the physical system described by the PDEs influences the structure of the matrix  $A$ .<sup>10</sup> Two types of procedures used to solve large sparse linear systems are *direct* methods and *iterative* methods. In general for large systems, iterative methods require less storage and may require fewer arithmetic operations than direct methods.

Iterative methods work by breaking up the system into subsystems. A number of different groupings are feasible: point row-wise, wavefront, submatrix etc. Let us consider some block iterative methods. Here the matrix  $A$  is parti-

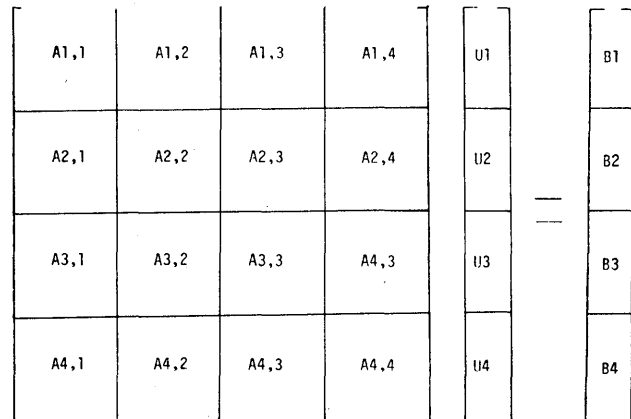


Figure 9—Linear system  $Au = b$  block partitioned with  $q = 4$ .

tioned into submatrices  $A(i,j)$  (Figure 9); the resulting subsystems are solved and the results are merged. The subsystems are often selected such that their solution is trivial, e.g. the submatrices may be lower triangular; if this is not the case other techniques can be used. This is repeated until some stopping criterion is fulfilled.

In the Jacobi method, one solves  $g$  subsystems during each iteration as follows:

- (2)  $A(i,i) * U_i(n+1) = -\sum_{j \neq i} A(i,j) * U_j(n) + B_i$  for all  $j \in NE.i$   
 $j = 1, q$   
 where  
 $A(i,j)$ : submatrix  $i,j$   
 $U_i(n)$ : solution subvector  $i$  at  $n$ th iteration,  
 $B_i$ : subvector  $B$ .

For a dense matrix or a sparse matrix without structure, for subsystem  $i$  to compute  $U_i(n+1)$  it must have all  $U_j(n)$  where  $j \in NE.i$ . Thus  $g-1$  transfers are required for computing a subvector and  $q^*(q-1)$  transfers are required in computing the result vector per iteration. However, a knowledge of the structure of matrix  $A$  can be used to reduce the number of transfers.

Consider the data flow graph of the iterations (Figure 10)—each node in the graph represents the solution of a subsystem; the directed arcs represent data transfers from a subsystem to another subsystem. Data must be present on all the input arcs at a subsystem before that solution can proceed. In this case  $q$  subsystems at the  $n$ th iteration must be solved before any of the solutions at the  $(n+1)$ th iteration can proceed. Again, a knowledge of the structure of matrix  $A$  can be used to permit those subsystems to proceed, whose input arcs are full.

Consider the case where each subsystem is assigned to a partition. Each partition can operate in one of 3 modes:

1. Computation mode: Intrapartition computation. This can involve intrapartition synchronous packet transmission, e.g., as in a matrix multiply.
2. Communicating mode: Receiving/transmitting data from/to another partition.
3. Idle: Not (1) or (2).

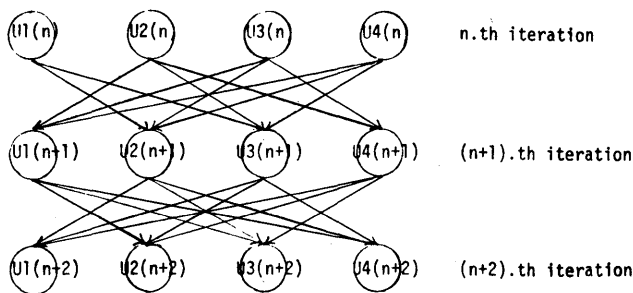


Figure 10—Data flow graph for Jacobi method.

The current version of TRAC does not support the overlapped operation of (1) and (2) although there is no intrinsic reason in the TRAC architecture which will prevent its extension to this further dimension of parallel execution.

For the Jacobi method the computation for an iteration of subsystem solution is according to equation (2). This computation can be performed using a matrix multiply and a vector add.

The Gauss Seidel method (which is structurally similar to several other methods) is a better technique on the basis of convergence criteria; it requires the solution of  $q$  subsystems of the form:

$$(3) \quad A(i,i).U_i(n+1) = -\sum_{j=1, j \neq i}^q A(i,j) * U_j(n+1) - \sum_{j=i+1, j \neq i}^q A(i,j) * U_j(n) + B_i$$

The corresponding data flow graph is shown in Figure 11. The dependencies in this graph indicate that only one node can be active at a given time. Prepaging techniques based on<sup>11</sup> can be realized using structures similar to that shown in Figure 12. This can result in up to 50 percent reduction in processing time.

An analysis of the dataflow within each node indicates that all the inputs are not necessary for the operation of the node to commence (Figure 13)! We can still assign a subsystem per partition, but by permitting each subsystem to proceed when it has sufficient input data, utilization is enhanced (Figure 14). This can give up to a  $(q-1)$  fold improvement on a  $q$  partition (with one subsystem per partition) system. We are currently studying these and other structures.

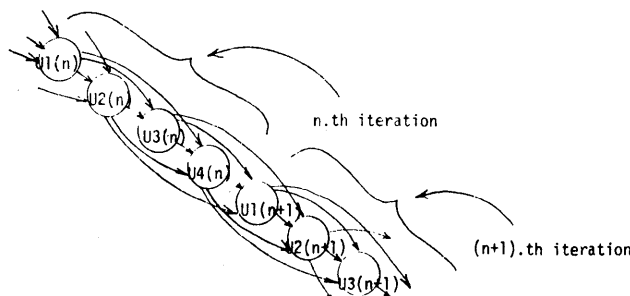
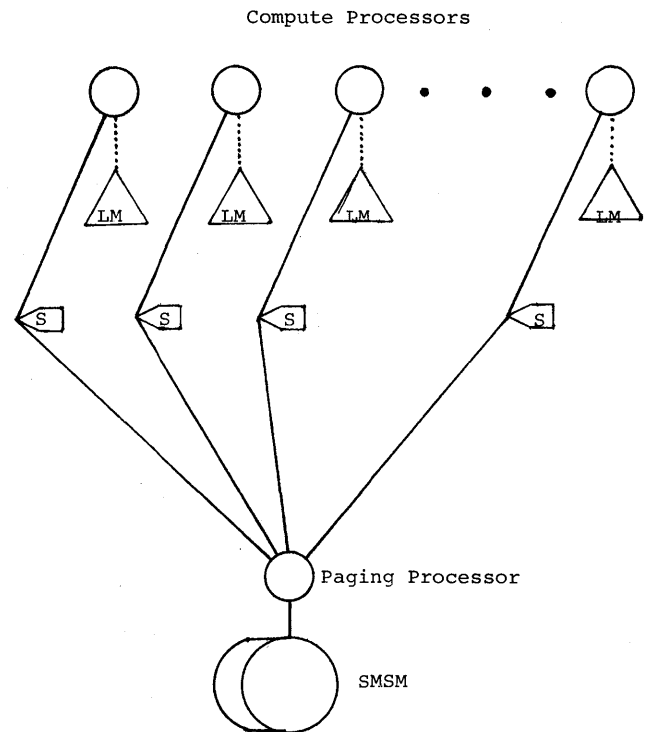


Figure 11—Data flow graph for Gauss Seidel method.



LM:Local Memory S:Shared Memory SMSM:Secondary memory

Figure 12—Structure for prepaged Gauss Seidel method.

### 2.2 Monte Carlo method

The Monte Carlo method consists of constructing a probabilistic model of a problem in which the problem solution is a statistical parameter of the model that is determined by repeated, independent random sampling. As with any statistical process, a large number of trials must be made in order to obtain accurate results. This method generally converges slowly as  $1/(\text{SQRT } n)$  where  $n$  is the number of trials performed (convergence acceleration techniques are known but are not included in this discussion). Because of this slow convergence, the Monte Carlo method is seldom used if the problem is amenable to other numerical techniques. A major advantage of this method however in terms of parallel processing, is that the solution at each point can be estimated independently (in parallel) and the method can easily be extended to higher dimensional problems with linear increase in computational complexity.

The Monte Carlo method was selected for evaluation on TRAC for the following reasons:

1. it is computationally intense;
2. it contains considerable inherent asynchronous parallelism;
3. its algorithms tend to be simple;
4. the method is useful in numerical applications.

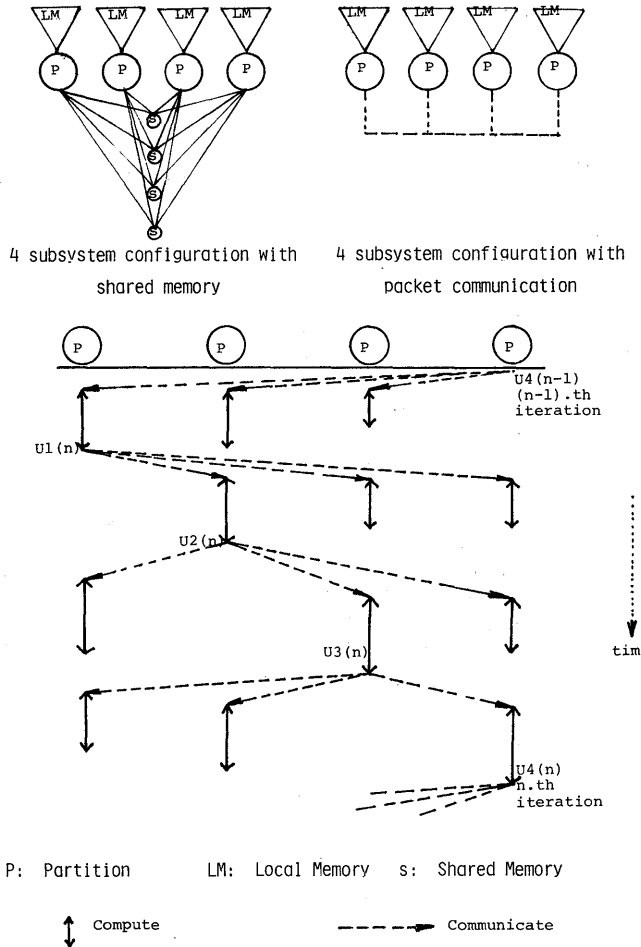
Initially a very simple numerical integration problem to find the area of a unit circle is used to benchmark TRAC. This problem, though extremely simple, contains the essential elements of the Monte Carlo technique. First, the mathematical problem is described, then an organization to compute the results that exploits as much inherent parallelism as possible is given. Finally, a mapping of this task structure onto TRAC is discussed including the potential performance of the system.

The mathematical model of the problem is shown in Figure 15 where the area has been normalized to the unit square for convenience. Generally speaking, one can describe a probabilistic model of the problem in which "darts" (represented by pseudo-random number pairs) are thrown at the square and the area of the circle determined by counting the number of times a dart hits inside the circle (successes) compared to the total number of throws. The area of the circle is then given by:

$$\text{AREA} = \text{HITS} / \text{TOTAL THROWS}.$$

As more and more "darts" are thrown the area of the circle is determined more accurately. Of course, the problem can be reduced by symmetry to consideration of the first quadrant only and the resulting area multiplied by four. Mathematically two independent sequences of random variables  $X_i$  and  $Y_i$  are generated. The points  $(X_i, Y_i)$  are tested to see if they lie within the circle by determining if  $X_i^2 + Y_i^2 < 1$ .

The steps of the algorithm can logically be partitioned between two task types which can run independently. Tasks



P: Partition LM: Local Memory s: Shared Memory  
 ↑ Compute      - - - - - Communicate

Figure 14—Timing for Gauss Seidel method solution.

of type A perform the calculations of the individual trials, and determine if the trial was a success (hit) or a failure. Tasks of type B check the prescribed convergence criteria and terminate the whole process when satisfied. Generally one would like to have a large number of type A tasks running in parallel and a single type B task checking their results.

The steps of tasks A and B are outlined as follows:

```

Task A
begin
  generate pseudo-random number pair  $(X_i, Y_i)$ 
  if  $X_i^2 + Y_i^2 < 1$ 
    success: = success + 1
    total: = total + 1
end
    
```

```

Task B
begin
  newarea: = success/total
  if  $(\text{newarea} - \text{oldarea}) < e$  stop.
  oldarea = newarea
end
    
```

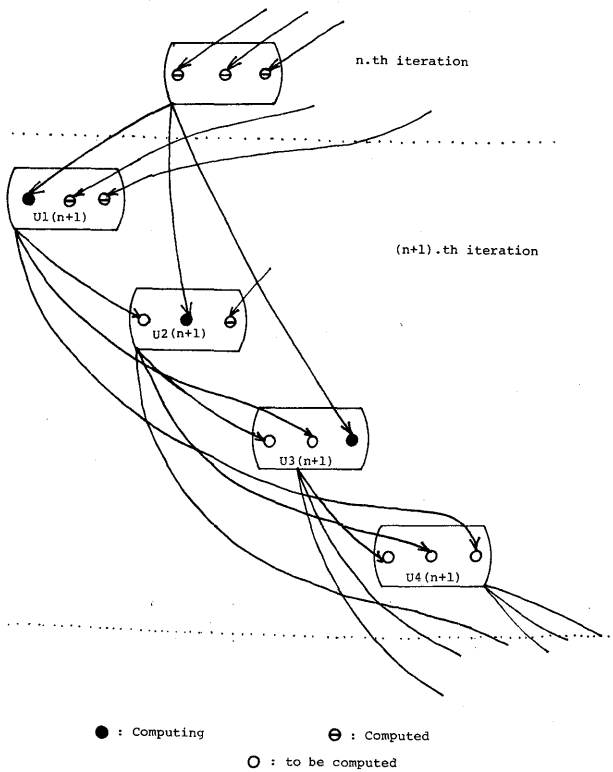


Figure 13—Dataflow within subsystem solution for Gauss Seidel method.

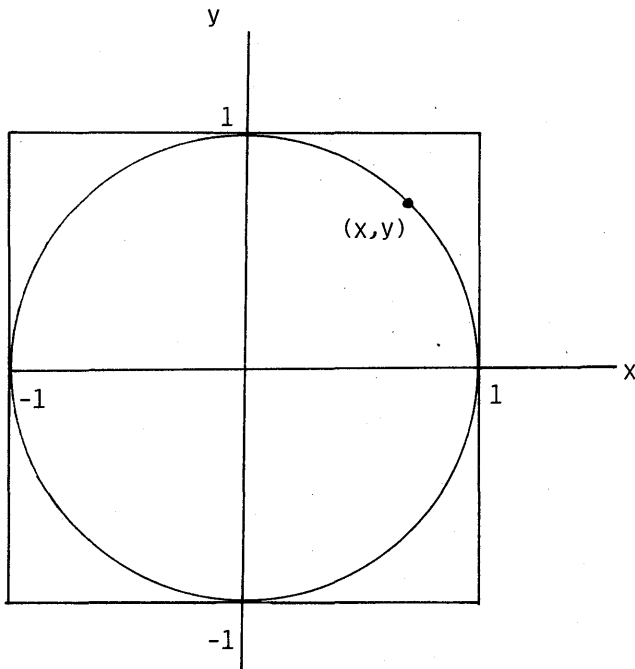


Figure 15—Monte Carlo integration.

A conceptual TRAC configuration is shown in Figure 16. Identical machines  $A_1, A_2, \dots, A_n$  would be configured and operate asynchronously reporting the results of their experiments by packets sent to task  $B$ . Task  $B$  would monitor the convergence and determine when to terminate the processing. Although this process organization fits the problem well and offers a potential reduction in processing time by very nearly  $1/n$  there are two problems:

1. Counter bottleneck. If  $n$  is very large (as one would hope for a full-scale TRAC machine), task  $B$  may become a bottleneck due to the large number of packets received from the  $A$  tasks. A possible solution is illustrated in Figure 17 in which a small number of additional  $B$  processes are introduced forming a tree to reduce the message traffic to each  $B_i$ . When a total counter in a  $B_i$  task overflows the next higher counter is updated. The updating process could be skewed to avoid bursts of updates caused by the  $A$  tasks running very nearly in lock step.
2. Parallel pseudo-random number generation. It was assumed that each task  $A_i$  generates its own pseudo-random numbers (PRN). This is a non-trivial assumption, since one must be sure that the parallel sequences of PRN are independent. A method has been developed

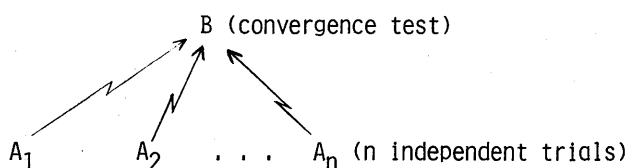


Figure 16—Centralized testing.

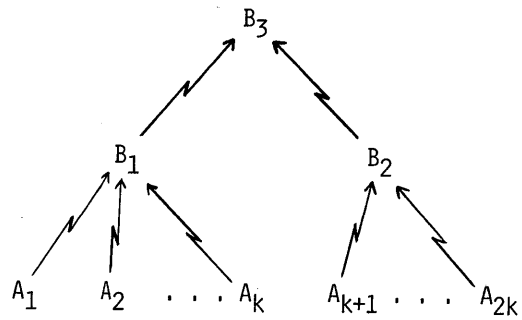
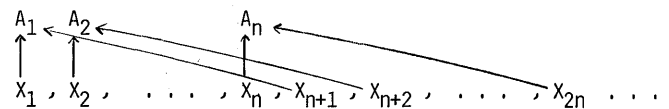


Figure 17—Distributed testing.

to guarantee that the sequences are independent. This method assumes a PRN sequence in which one has confidence. Given this sequence generated by a multiplicative PRN algorithm then each  $A_i$  is assigned a seed from the assumed PRN sequence (see Figure 18). An offset constant,  $c$ , is used by each  $A_i$  to generate its next PRN. More details and extensions of this method are found in<sup>12</sup>. Once the  $n$  seeds,  $X_i$  and  $c$  are computed and sent to the corresponding  $A_i$  tasks, each task can generate its PRN's in parallel independent of the other tasks.

### 2.3 Database management

The design and implementation of special machine architectures for database management has been receiving increasing attention in recent years. A major reason for this



Assume: A pseudo-random number generator of the form

$$X_{m+1} = (a \times x_m)_b$$

Process: Each  $A_i$  given seed  $X_i$  generates the PRN sequence  $X_i, X_{n+1}, X_{2n+1}, \dots$

Each PRN used by  $A_i$  is determined as follows:

$$\text{next PRN} = (c(\text{last PRN}))_b$$

$a$  : prime number used to generate seeds

$b$  : modulo base

$c$  : multiplicative constant offset,

$$c = (a^n x_i)_b$$

$n$  : number of parallel tasks  $A$

$X_i$  : seed used by process  $A_i$

Figure 18—Parallel pseudo-random number generation.

interest is the rapidly growing size of databases and the high frequency of query processing. Hsiao<sup>13</sup> in his description of data utilities, suggests a future database capacity requirement of over  $10^{12}$  bytes and a peak query frequency of a million requests/second.

A number of different approaches have been taken toward solving the very large database processing problem.<sup>14, 15, 16</sup> Reconfigurable machines are becoming feasible primarily due to the appearance of low cost microprocessors. The flexibility of such machines may permit a synthesis of the previously described database machine approaches with the additional capability for highly efficient concurrent query processing through configuration to match individual or groups of queries. In effect, special purpose sub-computers can be created to process each query and a number of these machines can run concurrently.

### 2.3.1 Approach

One approach suitable for the hierarchical data base management system is to consider TRAC as a potential backend machine to process indices to a large data file. The database data file will reside on a general purpose host system on a conventional mass storage device with secondary index files (inverted file and record association file) being searched and updated in the the backend, and implemented so as to take advantage of the backend's novel architectural features. A key file consisting of a key type, keys and a trace (logical identifier) for each record occurrence in the database is maintained on the backend. Traces described by Lowenthal<sup>17</sup> are tuples which represent record occurrences and the logical positioning of a record in a hierarchically structured database. Operations on the key file are functionally equivalent to operations on an inverted file. This file may be segmented by key type and paged into fast access SMSM memories<sup>4</sup> and searched to determine a set of traces which are then passed along to a set operation pipeline where they are combined with other trace sets to generate a response set for complex queries. Queries may be represented in a disjunctive form:

$$A(1).op.A(2).op. \dots A(i) \dots .op. A(n)$$

where each  $A(i)$  represents conjunctions of relational expressions defined on key items and values within a single record type and ".op." represents a set operation, such as union, intersection and difference. For example, consider the query:

Find name SMITH where salary is greater than \$20,000 and location is Austin.

Where name, salary and location are key items. Suppose each of these keys are on separate key files, then

$A(1)$ : (name, =, SMITH)  
 $A(2)$ : (salary, >, 20000)  
 $A(3)$ : (location, =, Austin)

and the query is of the form:

(name, =, SMITH) .and. (salary, >, 20000) .and. (location, =, Austin).

Processors are configured to search the corresponding key file for each of the  $A(i)$ 's. The  $A(i)$ 's have been constructed as simple operations so that a single pass through the file is sufficient to determine the response set. As a result of these searches, the average data traffic through the pipeline is greatly reduced.

A principal focus of this investigation is the development of efficient algorithms for sorting and set operations on indices. Efficient set operation algorithms are necessary to combine trace sets to determine the response for complex queries. Sorting is required to arrange the key file by trace value so that each trace set entering the pipeline will be sorted to facilitate set operations for individual queries. Sorting could be performed during slack periods or during updating sessions and could use the large number of processors that would be available at that time. Separating the sorting from individual query processing eliminates much redundant sorting.

While other architectures also offer potential for very large data base management, the inherent parallel but flexible TRAC structure promises to be a viable alternative to, or a useful addition to, those other techniques.

### 2.3.2 System configuration

Based on the approach described in the previous section the backend system is configured into multiple pipelines of the type shown in Figure 19. For the set operation stages it is assumed that in general the size of the trace sets will be large relative to the memory available for individual pipelines. Furthermore, traces will only be available page by page from the search stage. Under these constraints a pipeline approach is valid. It is expected that on the average several hundred pipelines could exist simultaneously on a machine with 6000 or more processing elements. Note that this is an average figure and the peak query concurrency could be higher.

High input bandwidth to the pipeline is required to avoid a possible I/O bottleneck. The approach taken is to page the key file, segmented by key type, from secondary memory to high speed SMSM. These memories output continuously, and the data can be read by attaching processors dynamically to the desired bytes of the key. Figure 20 shows an example input. The key portion and trace are read in parallel by the attached processors. Key bytes are compared with a comparand register slice in each processor, and if the query

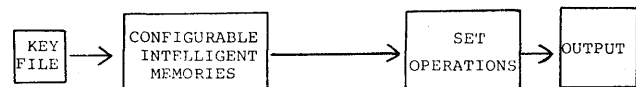


Figure 19—System functional pipeline.

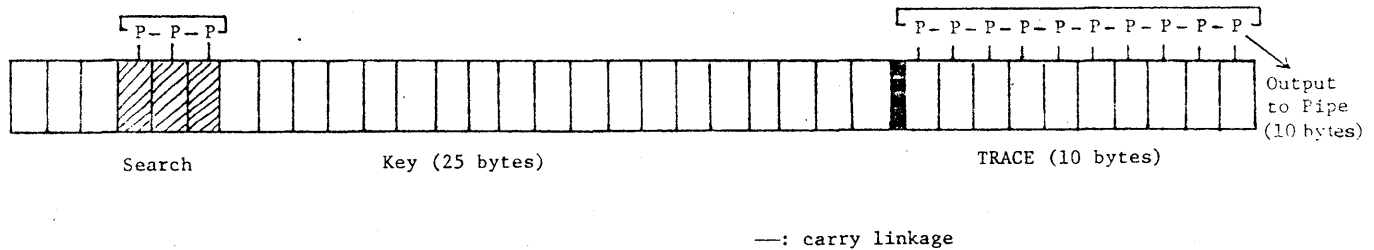


Figure 20—Pipeline input.

condition is satisfied, the associated trace is pushed into RAM buffers attached to the proper pipeline. More than one processor can be attached at the same time to a byte of memory so that a number of pipelines can be serviced concurrently by the same input module.

There is also the possibility of overlapping or time sharing stages of the pipeline. For example, set intersection may result in diminished output while set union will result in increased output. The flow through the network, therefore may not be smooth resulting in idle time that could be shared with other processes. Resources can be released as the pipeline begins to empty.

The interesting possibility of performance improvement through query lookahead and Boolean tree height reduction for optimizing the configuration and scheduling of the query pipelines is under investigation. Algorithms discussed by Kuck[18] and Ramamoorthy[19] for tree height reduction in arithmetic expressions may be applicable in this context.

### 3.0 CONCLUSIONS

TRAC offers the potential for a great amount of parallelism. We are studying the approaches to best utilize the computer, and preliminary results are positive. This paper summarized the architecture and its uses. The following papers on TRAC in these proceedings show more detail on how the hardware is built. Though not presented in depth in these proceedings, a joint effort between the University of Texas Electrical Engineering Department and the Computer Science Department has yielded extensive research and development in the areas of operating systems, system simulation and other software areas.

### 4.0 ACKNOWLEDGMENTS

The authors would like to gratefully acknowledge J. C. Browne, M. M. Malek, H. L. Taylor and the members of the TRAC team whose extensive works are reflected in this paper.

### BIBLIOGRAPHY

1. Lipovski, G. J. and Tripathi, A., "A Reconfigurable Varistruure Array Processor," *Proc. of the 1977 International Conference on Parallel Processing*, August 1977, pp. 165-174.
2. Wulf, W. A. and Bell, C. G., "C.mmp—A Multi-Miniprocessor," *AFIPS Proceedings*, Vol. 41, FJCC, 1972, pp. 122-131.
3. Goke, R. and Lipovski, G. J., "Banyan Networks for Partitioning on Multiprocessor Systems," *Proceedings of the First Annual Symposium on Computer Architecture*, 1973, pp. 21-30.
4. Lipovski, G. J., Su, S. Y., and Watson, J. K., "A Self-Managing Secondary Memory System," *Proceedings of the Third Annual Symposium on Computer Architecture*, January 1976.
5. Premkumar, U. V., "TRAC: Principles of Operation," Technical Report TRAC-3, University of Texas at Austin, January 15, 1979.
6. Smullen, J. R., "Memory Management of TRAC," Technical Report TRAC-2, University of Texas at Austin, May 1979.
7. Tripathi, A. R. and Lipovski, G. J., "Packet Switching in Banyan Networks," *Symposium on Computer Architecture*, 1979.
8. DeGroot, R. D., "Introduction to the Architecture of TRAC," Technical Report TRAC-1, University of Texas at Austin, January 1977.
9. Young, D. M. and Gregory, T., *A Survey of Numerical Mathematics*, Vol. II, Academic Press, 1972.
10. Young, D. M., *Iterative Techniques in Numerical Analysis*, Academic Press, 1971.
11. Trivedi, K. S., "An Analysis of Prepaging," CS-1977-7, Department of Computer Science, Duke University, August 1977.
12. Upchurch, E. T. and Lipovski, G. J., "Parallel Pseudorandom Number Generation and Monte Carlo Methods on TRAC," TRAC Technical Report, in preparation.
13. Hsiao, D. X. and Madnick, D. K., "Data Base Machine Architecture in the Context of Information Technology Revolution," *Proceedings, Third Very Large Database Conference*, October 1977.
14. Berra, P. B., "Recent Developments in Data Base and Information Retrieval Hardware and Architecture," *COMPSAC*, November 1978, pp. 698-703.
15. Dewitt, D. J., "DIRECT- A Multiprocessor Organization Supporting Relational Data Base Management Systems," *Proceedings of the Fifth Symposium on Computer Architecture*, April 1976.
16. Lipovski, G. J., "Architectural Features of C'ASSM: A Context Addressed Segment Sequential Memory," *Proceedings of the Fifth Annual Symposium on Computer Architecture*, April 1978, pp. 31-38.
17. Lowenthal, E. I., "A Functional Approach to the Design of Storage Structures for Generalized Data Management Systems," Ph.D. Dissertation, Computer Science Department, University of Texas at Austin, August 1971.
18. Kuck, D. J., et al., "On the Number of Operations Simultaneously Executable in FORTRAN-like Programs and their Resulting Speedup," *IEEE Transactions on Computers*, December 1972, pp. 1293-1310.
19. Ramamoorthy, C. V., et al., "Compilation Techniques for Recognition of Parallel Processable Tasks in Arithmetic Expressions," *IEEE Transactions on Computers*, November 1973, pp. 986-998.



# Design and implementation of the banyan interconnection network in TRAC\*

by U. V. PREMKUMAR, R. KAPUR, M. MALEK, G. J. LIPOVSKI, and P. HORNE

University of Texas  
Austin, Texas

## 1.0 INTRODUCTION

Over the past few years, owing to technological breakthroughs in building cheap, reliable and powerful microprocessors and relatively cheap LSI memories, interconnection networks have become the major hardware cost in design and implementation of the multiprocessor systems. This situation occurs from the fact that many more functions may be expected from the interconnection network (switch) than the establishment of simple bus connections. Even if only the communication links were considered, the complexity of some networks make their implementation prohibitive. An example of such a network is a crossbar whose complexity is  $O(n^2)$  where  $n$  represents a number of resources which may be connected to another set of  $n$  resources. This switch provides a separate connection between each pair of resources (Figure 1). It has been empirically shown that implementation of a crossbar switch for a large  $n$  is very difficult and with a state-of-the-art technology practically infeasible for  $n > 50$ .

In a multiprocessor system, the interconnection network is an expensive but essential component. The switch, which is central to the design of TRAC architecture, has been designed to achieve maximum capability within the limitations of LSI technology (low pin count and large gate count per chip).

Among several interconnection networks proposed in literature [1-14], banyans seem to be one of the most general classes and it has been proven that the majority of existing networks are special cases of banyans [7,13]. We might observe that several special cases of banyans have been described in the literature. A cross-point switch is a regular banyan of height  $l$ , and a  $(f,s,l)$  SW-banyan can be defined as  $l$  recursions on an  $f$  by  $s$  cross-point. An Omega network [5,6] is an  $f=2, s=2$  SW-banyan, a perfect shuffle [11] is a homomorphic reduction (i.e. the  $l$  layers are folded into each other) of an  $f=2, s=2$  SW-banyan, and the fast Fourier transform interconnection structure is an  $f=2, s=2$  SW-banyan.

While cross-point switches are indeed banyans, they are among the least efficient banyans and other banyan switches

have better cost performance. Their complexity as a function of resources to be connected grows in proportion to  $n \log n$ . Banyans can be easily controlled, they are flexible, and adaptable to specific applications by proper selection of the fanout and spread—parameters of the switch.

Preliminary theoretical analysis and simulation indicate favorable performance and small probability of blockage. The TRAC system has a banyan switch with  $f=3, s=2$  and  $l=4$  (Figure 2) as the interconnection network and provides a reconfigurable and varistructured architecture that supports SIMD and MIMD modes of operation.

## 2.0 SWITCH PERFORMANCE

Requirements imposed on the interconnection network for a reconfigurable machine may be very diverse. Reconfigurable systems naturally imply feasibility of multiple modes of operation which in effect stipulates arbitrary configurations of the system where processors, memories and I/O's can be connected in several ways, e.g. one processor to many memories (data trees), one memory to several processors (instruction or shared memory trees). Trees may be looked upon as communication busses connecting a given set of resources for SISD or SIMD operation. There may be several tree structures in the system at the same time. All setups should be controlled externally because individual control of each switching element is expensive and complicated. Varistructured systems should be capable of multiprecision arithmetic. This implies provision of busses for high speed carry propagation between an array of processors which process a single multiprecision word.

Shared memories must have arbitration circuits in order to ensure access to a shared memory by one processor at a time. Additional priority selection logic has to be designed for selecting a single processor from a set of candidate processors which are capable of connecting a specified set of memories. Similar circuits should provide for memory selection when a set of processors requires connection over a common memory.

A switching structure may perform several other functions such as synchronization of processor and memory cycles and packet switching. All of the above functions have been

\* This research is supported by RADC F30602-78-C-0099.



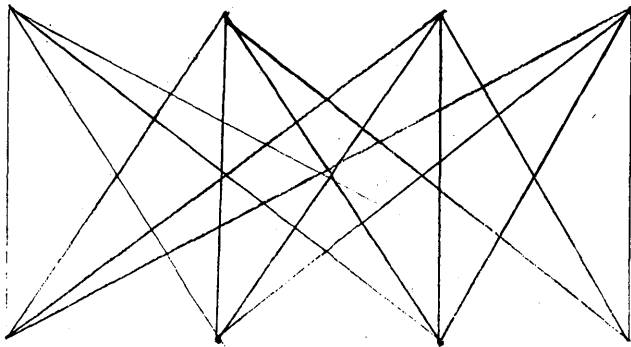


Figure 1—4 × 4 crossbar.

designed in the TRAC system and their implementation will be discussed later. We should also point out that besides the functions which we have mentioned earlier, we also expect a switching network to be cost-effective, capable of operating at high speeds, modular, expandable and fail-soft or fault tolerant.

Since complexity of banyan networks is  $O(n \log n)$ , it makes them definitely more cost-efficient than cross-points (for  $n \geq 10$ ). The majority of useful interconnection networks have complexity equal to or greater than banyans.

Delay, which has a direct impact on speed of operation, is an implementation criterion and is technology dependent. Its growth rate is  $O(\log n)$  as in the majority of multistage networks. Practically every multistage network may be constructed from a set of the same switching elements, therefore modularity is evident. A well designed system should have an expandable switch so that it would be possible to attach additional resources if required without discarding or completely rewiring the original switch.

Fail-softness implies that an interconnection network should be formed from fault independent units in order to protect a system from catastrophic failure. If part of the switch fails, the switch should be capable of connecting most of the resources, bypassing the failed part.

Fault tolerance is a highly desirable feature in ultrareliable systems where the probability of failure should be minimized and a system should always be capable of performing at maximum possible capacity. To ensure this, the system

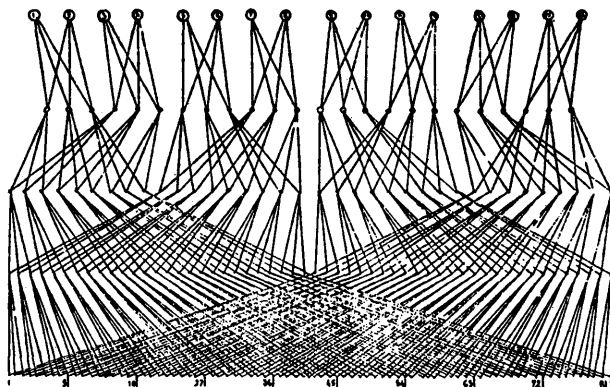


Figure 2— $s=2, f=3, l=4$  SW-Banyan network.

should have fast, automatic procedures for error detection and standby switching elements to replace the faulty ones.

Considering the above listed switch requirements and performance criteria, banyans seem to be very promising. These networks permit economical partitioning of resources in large modular systems into a wide variety of subsystems. They further provide cost-effective mechanisms for sharing of resources and for communication among subsystems. Inherent fail-softness capability, LSI compatibility and the existence of fast control algorithms which can be largely performed by distributed logic within the network are also important attributes of banyans.

### 3.0 FUNCTIONAL DESCRIPTION OF THE TRAC INTERCONNECTION NETWORK

The interconnection network or switch is the fundamental component of the TRAC architecture and provides a multitude of functions that support a reconfigurable, varistructured multiprocessor system with multiple modes of operation. In this section, we present some of the primary functions realized by the network and how they can be used for a wide variety of applications. The switch permits efficient means of setting up a spectrum of configurations. The network also supports a comprehensive communication among the processors via packet communication.

The TRAC interconnection network is a multistage network connecting a set of processors to a set of memories and/or I/O devices (MIO modules). The network consists of multiple levels and operates under the control of a system clock. The clock cycle is made up of several phases intended for different functions. The network supports bidirectional

TABLE I.

	120ns ø0	120ns ø1	240ns ø2	240ns ø3	240ns ø4	ø5
TR	Enable I & S trees			Copy S + I (Clock)	Destroy I&D Trees	
SL	clear AC FFS		clear S FFS	S → I, set AC shape priority circuit		
RF	missing page interrupt		false	data bus direction		
RB	missing page interrupt		available	data bus direction		
TL	available		setup carry direction	data bus direction		
DR	carry direction			carry (default) direction		
DATA BUS	sync pkts	async pkts	memcommd	data		
GR	Set I, D, & S FFS			available		
RQ	Set I, D, & S FFS			available		
DD	Data Tree denial				dist. interrpt	
DI	Data Tree denial					
CP,CG,CC	carry circuit			priority circuit		
PR,PG				sync pkt request	async pkt request	
SWCMD	clock for I, D, & S FF setup				reset	
A	0	0	0	1	1	0
B	0	1	1	1	0	0
C	1	1	0	0	0	0

buss connections among processors and MIOs. These connections may be quickly established with the use of hardware algorithms executed by hardware in the network via an external switch controller. In a given memory cycle, the switch acts like a set of (tree-shaped) wire-OR buses for half the cycle, and like a packet switching store-and-forward network in the other half of the cycle. This is done utilizing the time slot in the system cycle during which bus connections are not needed owing to the inherent memory access delays. Thus packet switching is implemented at relatively small additional cost.

The first subsection describes a variety of tree connections the switch is capable of configuring and their primary uses. The second subsection deals with the packet movement and resulting communication. The last subsection is devoted to the description of switch control.

### 3.1 Tree connections

The banyan switch is capable of setting up three basic tree structures, the data tree, the instruction tree and the shared memory tree [15]. A data tree connects a single processor to several MIOs and supports an SISD mode of operation. Data trees have leaves at the base nodes and roots at the apex nodes of the banyan (see Figure 2). The processors that form the root nodes of several data trees may be connected together to form leaf nodes of either an instruction tree or a shared tree. The configuration with an instruction tree can support a multiprecision SIMD mode of operation. Instruction or shared memory trees have leaves at the apex nodes and roots at the base nodes of the banyan (see Figure 2). The configuration with a shared memory tree can support an MIMD mode of operation. It is possible to switch a shared tree into an instruction tree and switch back. This permits the SMIMD mode of operation [16] by selectively and dynamically creating and destroying instruction trees from existing shared trees. Embedded within either an instruction tree or a shared tree, there exists a General Purpose Communication link.

#### 3.1.1 General purpose communications link

Carry-lookahead circuits can be used for more than just adders! The use of carry lookahead logic as a general purpose communication (GPC) link has been described in [17]. A slight modification of the scheme is used in the TRAC system to achieve a flexible one-bit bidirectional data transfer among a group of processors connected by an instruction tree or a shared tree whenever such a tree is created. (Pieces of GPC links exist in every node in the banyan, and are connected into the GPC link when the instruction tree is created.) The GPC link is embedded in the instruction tree. A schematic diagram of a GPC link consisting of three levels and connecting eight processors is shown in Figure 3.

Each node in the GPC link contains the carry lookahead logic required to connect its one or two sons to its father. The GPC link thus provides a high speed one-bit commu-

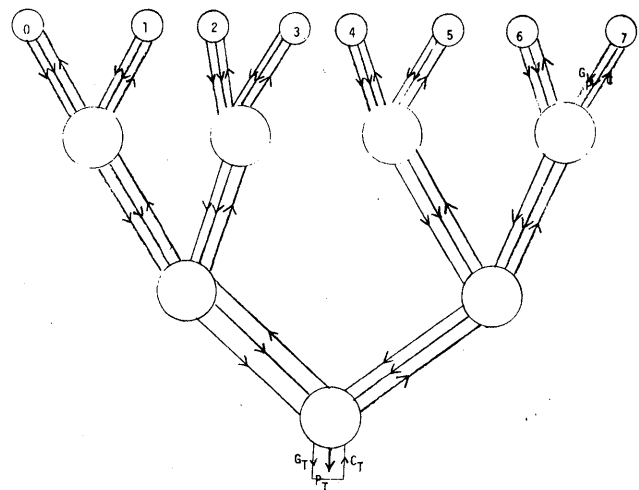


Figure 3—GPC link.

nication link on which the delays of propagation are proportional to the number of levels in the network. Each node in the GPC link has the capability to reverse the linking order of the processors connected by its subtree. This feature enables flexibility in achieving a desired order of processors within a GPC link and is essential in configuring tasks consisting of fixed memory or I/O modules.

Each link in the GPC link consists of four distinct lines. The direction control line (DR) is used to determine the direction of data transfer within the ordered set of processors, i.e. forward or reverse. For example, left shift requires a forward and right shift requires a reverse direction of propagation. The other three lines G, P and C have their usual significance within the context of a carry lookahead logic. Depending on the way the G and C lines are connected in the root node, the processors would be either linearly or circularly ordered. For example, linear ordering is used for 2's complement adders and circular ordering is used in the "end-around carry" ones complement adder. Thus the GPC link provides a piece-wise or global data transfer in terms of broadcast, collection, priority and carry lookahead linkages among a set of linearly or circularly ordered processors. The use of this flexible communication scheme is described in [17].

#### 3.1.2 Shared trees

Shared memory trees are structurally similar to instruction trees: a shared memory tree is rooted in a memory module at the base of the switch and has two or more processors for its leaves. The links of the shared tree are, however, only potential connections that have been reserved for the configuration. Any one of the processors on a shared tree can acquire the shared MIO if it is not already held by some other processor on the shared tree when the request is made. An acquisition results in the establishment of an *active chain* between the shared memory and the processor that owns it, and the acquired MIO and the active chain become part of the acquiring processor's data tree. An explicit release op-

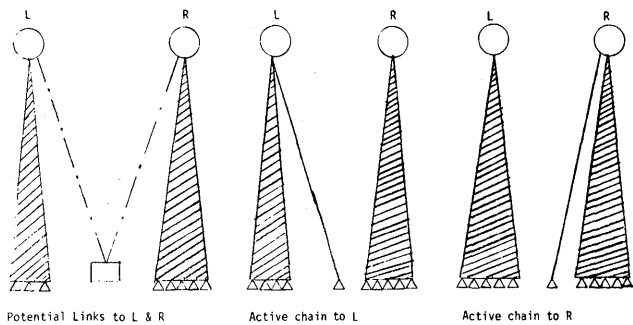


Figure 4—Shared memory: potential chains and active chains.

eration converts the active chain back to a potential chain and the shared MIO to an available MIO (Figure 4).

More than one processor can attempt to acquire an MIO at a given instant. These requests are arbitrated by a GPC link that is embedded in the shared tree (this uses physically the same components as the GPC link that follows the instruction tree but this GPC link follows a shared memory tree). The GPC link is used as a prioritization network that assigns a linear ordering to the processors at leaves of the shared tree. Mutual exclusion of acquisition is assured by the GPC hardware.

When configurations consisting of more than one shared tree between a set of processors are established two situations can result:

1. One processor has the highest priority in all shared trees, or
2. Different processors have highest priority in different trees.

The first situation results in unfair acquisition: if the highest priority processor makes a request it acquires all shared MIO's that are available. The second situation results in an undefined (deadlock) state. This is resolved by dividing shared MIO into distinct classes or colors. Arbitration for differently colored trees is performed in different memory cycles on the GPC link (i.e., eight colors are implemented in TRAC: a shared memory tree of color zero can be set up every eighth memory cycle).

A processor can belong in more than one shared tree—however, requests for shared MIO acquisition are made at different times as determined by the color. This can avoid deadlock if shared memory trees between a processor and each other processor that shares memory with it are of different colors (as in the graph theoretic coloring problem). Additionally, different processors can have highest priority in differently colored shared tree. This way one processor does not hog all the shared resources.

### 3.2 Packet switching

Packet switching provides communication from every processor in the machine to every other processor. This is achieved by making the entire switch available for packet movement in the time slot when tree-shaped busses are not

needed. This time slot is sufficiently wide for the time multiplexed implementation of two packet networks: these networks are called the mapping and the interrupting packet networks. The motivation for the implementation of two networks is discussed elsewhere [18].

The operation of all memory modules is synchronized so that all modules perform memory accesses in the same time slot (Figure 5). Trees are unnecessary when this access occurs—we use this time slot for moving packets from level to level in the switch. This is in keeping with the philosophy of extending switch functions at small incremental costs; further, the performance of the existing switch is not degraded in any way. The additional hardware needed for packet switching are buffers on switch links (one for mapping packets and one for interrupting packets per link) and arbitration logic on switch nodes (common hardware for mapping and interrupting packets).

Each data tree has one designated memory module. Packets are sent from this base node toward the addressed apex node of the banyan. A unique path exists from every memory module to every processor. This path is defined by a  $l$  digit address in base  $s$  notation (for regular banyans), where  $l$  is the number of levels and  $s$  the spread of the banyan (Figure 5). Each packet consists of two parts: a destination address and data. Packets in the current version of TRAC are 4 bytes long: 1 byte for address and 3 for data.

In the following discussion we will only consider one packet network. The behavior of the other network is identical.

Each link contains a single byte buffer. Packets move as byte trains from level to level wherever a buffer in a suitable link above the header becomes available. The choice of the

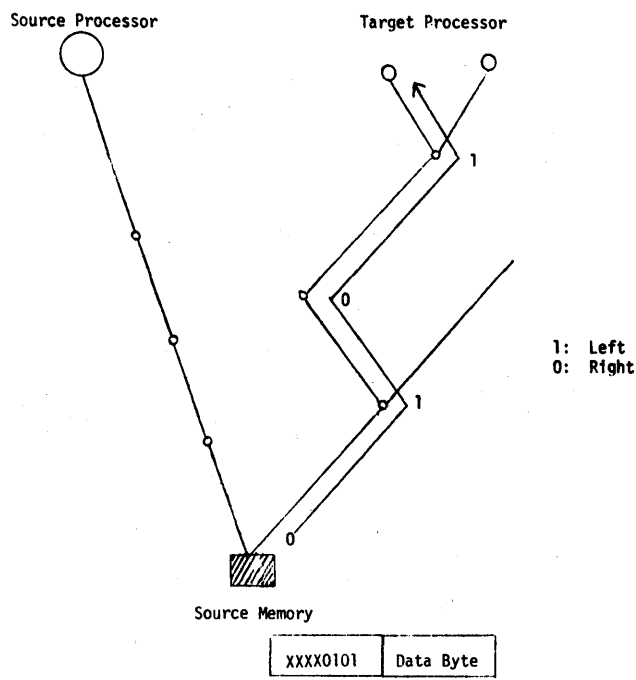


Figure 5—Packet movement on  $s=2, l=4$  banyan.

link is made on the basis of an appropriate digit of the destination header. All four bytes of the train move together up one level in a memory cycle.

Packet trains are synchronized. Every fourth memory cycle, the first byte of a train may enter the network from the base node. If trains will contend at a node, they will contend when their first bytes meet at that node. If more than one packet attempts to use the same node in the switch for jumping from link to link, prioritization hardware at the node chooses one (first byte) of the packets for movement and allows the remaining packets of the train to move in later cycles, while the other packets must wait (Figure 6). This results in nondeterministic transmission times for packets—however, no packet gets lost in the switch.

All processors can be transmitting packets in parallel. High data throughput is possible because a very large number of the links can be involved in data movement.

### 3.3 Master switch controller

The switch has a controller that is used to create trees. This controller receives instructions from a base node in the banyan. A *scheduler* is one of the tasks running in TRAC which includes, as a resource, this control port. The scheduler allocates resources for tasks, and issues commands to connect the resources. Commands are sent to the control port just like data might be sent to an MIO. Upon powering up the machine, the scheduler itself is set up, by reading a sequence of commands from a ROM in the control port. As trees are set up to avoid using faulty nodes, the scheduler is created from good resources. The only single failure point is the control port, thus enhancing reliability.

Processors have a physical address, called the processor address, which is their position at the apex. (This address is the same one used in packet switching.) In our prototype, it is a 4 bit binary number. Memories and I/O devices have a similar physical address, called the MIO address which is their position at the base. The resource address, in our prototype, is an 8 bit binary coded ternary number with a don't care digit code, in the manner suggested by Siegel [9]. If an address is 1,0,2,X, where X is a don't care, three base nodes,

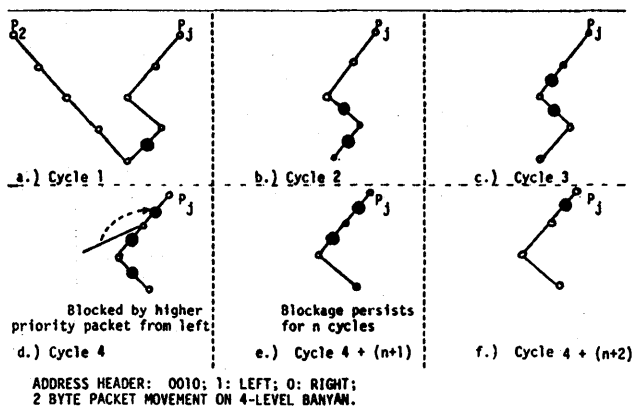


Figure 6

1,0,2,0, and 1,0,2,1 and 1,0,2,2 are selected at once. Don't cares speed up the selection of resources and tend to assign them so that the banyan is more efficiently utilized.

A task is a program segment that utilizes the same or similar data structures such as a subroutine to invert a matrix. A task may require  $n$  processors, each processor requiring  $m$  memory and I/O devices. To set up a task, a sequence of commands will be sent via the control port to set up busses (trees) in the banyan to interconnect the resources needed for the task. Data busses are first set up. A data buss is created by naming the MIO addresses in a command to mark them to be connected. The command includes a parameter which is the list of MIO addresses. MIO's are specified this way; however the processor is not specified. The processor is selected automatically by hardware, to avoid faulty or busy nodes in the banyan. The command returns the address of the processor that was selected by hardware through the control port, back to the scheduler, so that the scheduler can use this address later to set up memory sharing busses, and so that it can pass the address to any processor that may want to send packets to the processor that was selected. A data bus is set up in this manner for each of the  $n$  processors. Then an instruction bus is set up by another command to connect the processors together. A memory sharing bus can be created to interconnect the processors of selected tasks. Yet another command will create a memory sharing bus among  $n$  processors whose processor addresses are specified in the command. It returns the resource address of the memory that was selected, so it can be identified later. Instruction and data busses used in a task are automatically deleted by the task when it is done. Memory sharing busses generally have to exist as long as any task may need them, so they have to be explicitly disconnected by the scheduler. Note that tasks are set up rather infrequently (say every 100 ms.) so that the trees are created infrequently and remain to be used for long periods of time.

### 4.0 SWITCH ARCHITECTURE

After studying several interconnection networks we have decided to design and implement SW-banyan due to its flexibility, ease of control, cost-effectiveness, small delay, modularity, expandability and ease of diagnosis and fail-softness.

Since the family of banyans is very large, a choice of banyan meeting our requirements was not trivial.

The TRAC system (Figure 7) consists of sixteen 8-bit, bit-sliced microprocessors which are connected through the banyan network (switch) to the 81 MIO's (64 RAM's, disk and 16 I/O ports). In our study in choosing the banyan, we made the following assumptions:

1. The system should have 16 processors,
2. The ratio of the number of MIO's to the number of processors should be 3:1 to 5:1,
3. The fanout and spread should be between 2:1 to 5:1 due to physical implementation (preferably not larger than 4),
4. Cost and blockage should be minimized.

We have simulated all feasible banyans with 16 apexes and 63 to 81 MIO's and we have decided to choose a banyan of  $f=3$  and  $s=2$  due to its relatively low cost, minimal blockage (note that  $f=3$  and  $s=2$  average to the closest set of  $f$  and  $s$  to  $e$ ), acceptable number of levels and ease of addressing.

The following comparison with crossbar switch may prove promising capabilities of the banyan switch.

	BANYAN	CROSSBAR
Complexity	$0(n \log n)$	$0(n^2)$
$f, s$	arbitrary	fixed
Addressing	easy	may be complex
Blockage	small	none
Setup	easy	easy

The picture is even more clear if we actually compare banyan versus crossbar for 16 processors, 81 MIO's system. The TRAC's banyan is shown in Figure 2.

	BANYAN	CROSSBAR
Fanout	3	81
Spread	2	16
Number of nodes	390	1296
Number of links	211	96

This network is used to partition resources to set up multiple processes in the system [19]. As we have described earlier, the resources in process domain are dynamically re-configured to establish data trees, instruction trees and shared memory trees to execute programs. The current cost of TTL components and PC card used for a node is \$125 which amounts to \$26,375 for this network (211 nodes  $\times$  \$125 per node = \$26,375). The speed considerations indicate relatively small time required for scheduling and resource allocation (about 200-300 microseconds). The setup time is very small (1 cycle = 1 microsec). The data rate may be high since delay over the switch is equal to 20ns, which amounts to the total of 100ns over 5 levels of the switch. Blockage seems to be small. Simulation runs have indicated that well over 90 percent of the jobs could be allocated in three tries and on the average about 35 percent in the first try. Since modularity, expandability and fault tolerance are inborn qualities of banyans, all of them may be also found in the TRAC banyan.

## 5.0 FUNCTIONS OF THE SWITCH AND CONTROL LINES SPECIFICATION

The switching network performs several functions which can be divided into five main selections[20]:

1. Clock decoder logic,
2. Bus control logic,
3. Link control logic,
4. Packet switching logic,
5. Carry lookahead/priority logic.

Each node-cell in the banyan is put exactly on one printed circuit board (module).

Banyan switch signals may be divided into two categories: out-going and in-going into the node-cell.

Twenty-six lines run from node to node corresponding to each link between them. Of these 26 lines per link, 23 are presently being utilized.

These 23 control signals are explained below. Arrow indicates whether that signal propagates from processors to MIO's ( $\downarrow$ ) or vice-versa ( $\uparrow$ ). Several functions are time multiplexed on each line. The control lines are named according to their primary functions.

The following switch control signals are used:

1.  $\overline{GR} : \downarrow$  : *Grant*-Used to Broadcast request for Instruction-tree or Grant for Data tree. Broadcast is universal.
2.  $RQ : \uparrow$  : *Request*-Used to Broadcast request for Data tree or Grant for Instruction tree. Broadcast is universal.
3.  $\overline{TR} : \downarrow$  : *Tree*-Follows any active node and links. Starting from top it goes to all three nodes below it but passes through only the one that is active.
4.  $\overline{DI} : \downarrow$  : *Deny Instruction*-If a RQ signal coming down the Grant line runs into an active node a DI is generated at that node which is propagated down to bottom-nodes.
5.  $\overline{DD} : \uparrow$  : *Deny Data*-If a RQ signal going up the line runs into an active node a DD is generated at that node which is propagated up to top nodes.
6.  $\overline{RF} : \uparrow$  : *Read From*-In response to a request from Processor following instruction tree if the page no. in memory matches, a RF signal is sent back up the RF line.
7.  $\overline{RB} : \downarrow$  : *Rebound*-In response to a RF signal from a matching Memory, the Processor sends back a Rebound signal, asking memory to send up the Data byte along the Data Bus.
8.  $\overline{IL} : \downarrow$  : *Instruction Link*-Sets up the direction of Carry in the Carry look ahead circuit, in addition to its primary function of informing the MIO that it has an Instruction tree. Also used for Data Bus Direction.
9.  $\overline{SL} : \uparrow$  : *Shared Link*-Follows the shared tree. Informs the Processor that it has a shared tree. Propagates after the shared tree has been established.

10.  $\overline{PR} : \uparrow$  : *Packet Request*-
11.  $\overline{PG} : \downarrow$  : *Packet Grant*-

Packet Request is sent up to the node immediately above requesting permission to move a packet. If PG, Packet Grant, is sent down the packet moves up one level.

12.  $\overline{CP} : \downarrow$  : *Carry Propagate*-
13.  $\overline{CG} : \downarrow$  : *Carry Generate*-
14.  $\overline{CC} : \uparrow$  : *Carry*-

CLA (Carry Look Ahead) circuit is used to link the Processor-carries between successive Processors. Also used to determine shared Memory Priority.

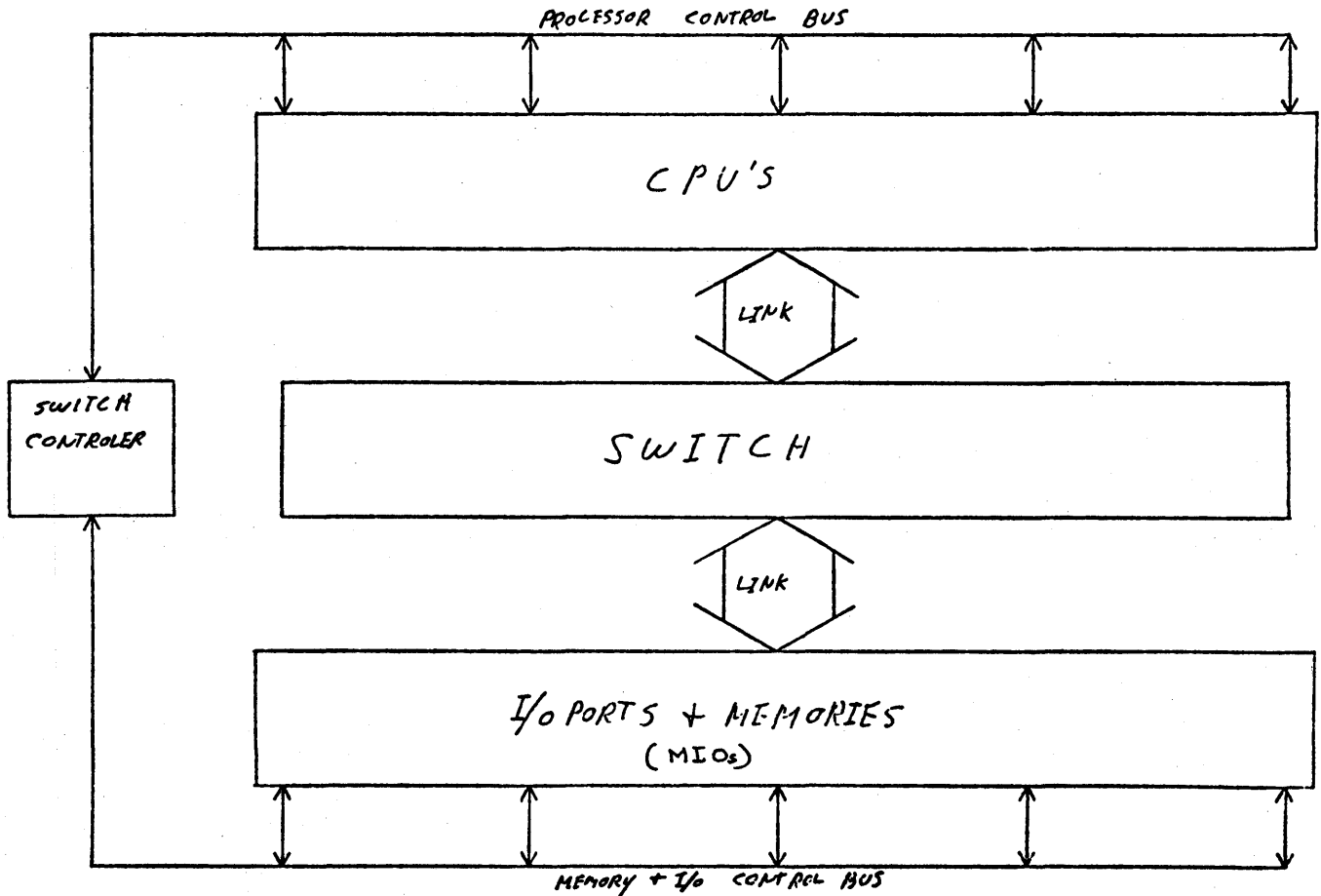


Figure 7—TRAC systems.

- 15.  $\overline{DR} : \downarrow$  : Direction-It provides a momentary change of direction in Carry circuit and Priority circuit.
- 16. DB  $\updownarrow$  : Bidirectional Bus for Instruction and Data byte transmission.

6.0 DESIGN AND DESCRIPTION OF THE BANYAN SWITCHING ELEMENT (NODE-CELL)

In this section the actual design of the banyan node-cell (with logic diagrams given at the end of this section) is introduced. The five functions of the banyan switching element (node-cell) and its implementation are described in detail in the following sections.

6.1 Clock decoder logic

Clock decoder enables timing of the switch with subcycles during which particular signals required for bus set up are generated. Inputs to this circuit come directly from the master control circuit. The inputs and outputs of this circuit can be classified as:

- (a) Switch command signals (SWCMD): Switch commands exercise absolute control over the type of tree

being set-up-*I*, *D* or *S*-type. The following table explains this 3-bit command.

SWCMD-0		
	SWCMD-1	
		SWCMD-2
0	0	0—No operation
0	0	1—Allows Data-tree set-up.
0	1	0—Allows Shared-tree set-up.
1	0	0—Resets the entire switch (during phase-4 only)

Further discussion pertinent to SWCMDs follows in Section 6.3 on link control logic.

- (b) Clock phase signals: A grey code generator generates A, B, C as shown in the circuit on clock decoder logic in Table I. These are used to generate the clock phase-signals  $\theta_0, \dots, \theta_5$ .
- (c) Packet switching signals: These signals are for exclusive use of the packet switching and discussion of these signals is deferred until the section on packet switching.

6.2 Bus control logic

The primary function of the bus control logic is to provide the flow of a byte on the bi-directional bus by enabling it in the proper direction and phases as required by the type of tree set-up.

Figure 8, a block diagram for the bus control logic, shows clearly the signals within this section.

During phases 0 and 1, the bi-directional bus is always off and the flip-flop bank in the bus control circuit is used for the packet movement only.

6.3 Link control logic

The following section summarizes the functions of the link control logic:

1. Set-up a Data-tree or deny it.
2. Set-up an Instruction tree or deny it.

3. Set-up a Shared tree or deny it.
4. Copy an *S*-tree to an *I*-tree.
5. Destroy an *S*-tree.
6. Destroy *I* and *D*-trees.
7. Reset the entire banyan.

(1) To set up a Data-tree:

Following steps must be executed for a Data-tree set-up.

- (a) SWCMD is set to 001.
- (b) MIO sends a request for the data-tree to the processors by setting the RQ-line. This broadcast is universal.

Note: If one of the *I*, *D* or *S*-ff is set, a *DD* (Data-tree denial) also accompanies the RQ-signal and the processor receiving both request and denial will not respond.

- (c) If the processor is available a grant is sent down by setting GR-line. Then during a phase 4, the *D* flip-flop is clocked and a Data tree is established.

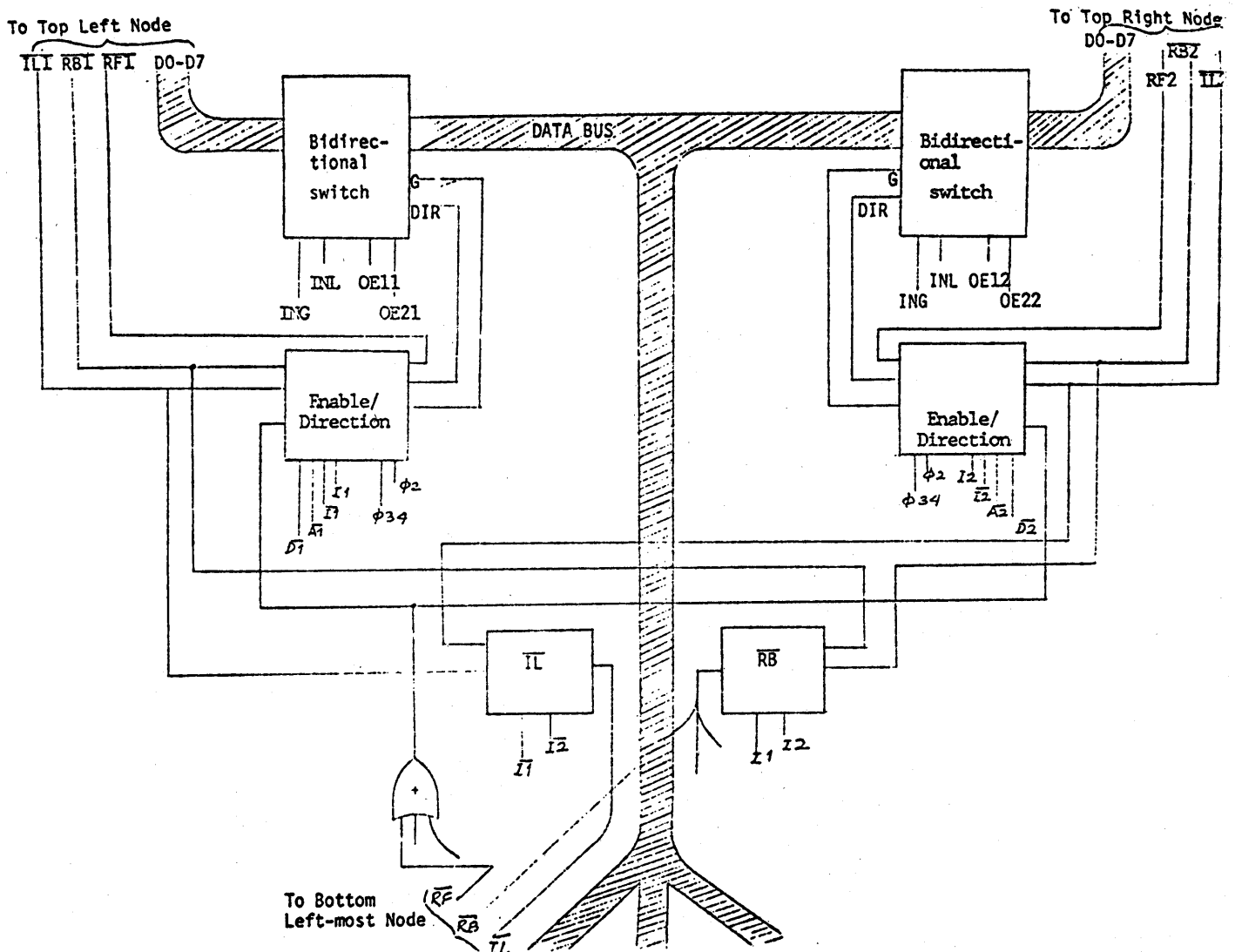


Figure 8—Block diagram for bus-control circuit.

Note: After a *D*-tree is set up a *DD* is now generated from this node and any future attempts to set a *D*-tree with this node without clearing *D* flip-flop will be denied.

(2) To set up an instruction-tree:  
Following set-ups must be executed for an Instruction-tree set-up.

- (a) SWCMD is set to 010.
- (b) Processor sends down a request by setting GR-line. Processor also sets TR-line to disenable DI signal being generated in the *D*-tree. This TR signal will follow only the *D*-tree, associated with this processor.

(c) After one or more of MIO's are chosen a grant is received by the node on RQ-line from the node below.

Note: The roles of RQ and GR line are interchanged in the case of *I*-tree set-up.

(d) After receiving a grant on RQ-line the processor clears the TR-line (must be done before phase 4).

(e) During phase 4 *I* flip-flop is clocked and an *I*-tree is set up.

Note: *D* flip-flop and *I* flip-flop in a node can be set, but not *set up*, concurrently.

(3) To set up a Shared tree:  
*S*-tree is set up in exactly the same way as an *I*-tree except the SWCMD in this case is 011.

(4) To copy an *S*-tree to an *I*-tree:  
Following steps must be executed for copying *S* to *I*-tree.

- (a) MIO sets the SL-line and processor sets TR-line.
- (b) Now if SL (*i*-1) is true, SL (*i*) is the value of *S* flip-flop in the node and during phase 3 in presence of TR signal it is clocked into *I* flip-flop.

Note: Clocking for entire *S*-tree must occur simultaneously in order for it to be copied correctly. Also TR-signal must become false before phase 4 or else it will clear the flip-flops in the node.

(5) To destroy an *S*-tree:  
To destroy an *S*-tree SL-line is set during phase 1 which generates S2C clearing *S* flip-flop.

(6) To destroy *I* and *D*-trees:  
TR-line is set during phase 4 which clears *D* and *I*-trees.

(7) To reset the entire switch:  
Set the SWCMD to 111 during phase 4. This clears the entire flip-flop set-up in the node cell.

6.4 Packet switching logic

The purpose of packet switching is to provide for a completely free movement of byte from any MIO to any processor. A packet then, by implication, can move from a MIO to a processor even if they are in two different trees concurrently.

Before explaining the packet switching logic it is necessary to understand the following:

- (a) Distribution of packet switching clock signals amongst different levels of the banyan.
- (b) Addressing for packet movement.

The clock is dispersed from level 1 through level 4. C(0), C(1)....., C(4) are connected to the system clock via a distributor network so that one system clock is sequentially transmitted to each of these. Each level interprets this cycle for one of the three purposes:

1. Negotiate (N): During the negotiate cycle this level is negotiated for by the level below.
2. Direction (D): If granted, the direction byte of the packet moves in during this cycle.
3. End (E): In this cycle, the last packet is moved out from this level. Also this cycle inhibits any grants to the level below even if the packet in this level has a grant to move up.

Example: C-2 is an END cycle for level 3, NEGOTIATE cycle for level 2, DIRECTION cycle for level 1. Therefore during cycle 2, a direction byte (if granted) moves into level 1 while level 2 is simultaneously being negotiated for and level 3 is moving (if granted) an END byte.

Figure 9 shows the packet movement at different levels in the switch. Figure 10 shows the block diagram for the entire packet switching circuit.

Note: Mapping and interrupting packet movement being mutually exclusive can share the priority circuitry.

Phase 3—Mapping packet negotiates.

Phase 0—Mapping packet moves.

Phase 4—Interrupting packet negotiates.

Phase 1—Interrupting packet moves.

Note also that the bus is used for the packet switching only during phases 0 and 1. During phases 3 and 4 only negotiation is in progress and that is completely restricted to packet switching circuit in the node and thus bus can be allocated for other purposes.

	T0	T1	T2	T3	T4	T5	T6
C P U's	---	---	---	NEG	DIR	1 <sup>st</sup> Data Word	2 <sup>nd</sup> Data Word
Top Node	---	---	NEG	DIR	1 <sup>st</sup> Data Word	2 <sup>nd</sup> Data Word	END
Center Node	---	NEG	DIR	1 <sup>st</sup> Data Word	2 <sup>nd</sup> Data Word	END	NEG
Lower Node	NEG	DIR	1 <sup>st</sup> Data Word	2 <sup>nd</sup> Data Word	END	NEG	DIR
Memory I/O	DIR	1 <sup>st</sup> Data Word	2 <sup>nd</sup> Data Word	END	NEG	DIR	1 <sup>st</sup> Data Word

Figure 9—Cycle chart, showing the movement of mapping or interrupting packet through the banyan.



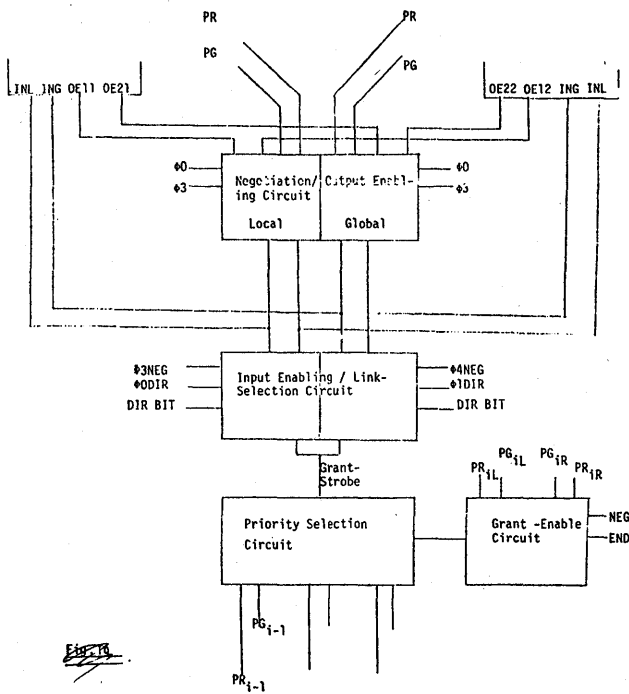


Figure 10—Block diagram for packet switching circuit.

6.5 Carry look ahead/priority logic

Carry look ahead/priority logic has the following two functions:

- (1) To provide a carry linkage between any two processors.
- (2) Priority selection for shared memories.

Carry linkage is enabled if it is a part of the Instruction-tree. Carry is used for the addition, subtraction, shift and the vector or element condition code transfer. A flip-flop is set when the shared-tree that it is part of is active. Figure 11 shows the carry-look-ahead circuitry.

6.6 Fabrication and testing

The switch module card contains mode logic and link logic for the two upper links. It is connected by two 26 wire cables to the modules above it and by three 26 wire cables to the modules below. A 12 pin edge connector supplies power and clock signals from the 'backplane.' The card itself contains 58 SSI and MSI chips. The bidirectional amplifiers are mounted on a piggyback carrier so that different bidirection amplifiers can be easily put in the module.

A microcomputer system with a 144 bit wide I/O port is used for testing the modules. For the switch module, all the 142 pins on the board are connected to the computer I/O

The  $G_x$ ,  $P_x$  and  $C_x$  of each I.C. connect to the appropriate connections on the memory modules. Each I.C. can handle 4 memory modules.

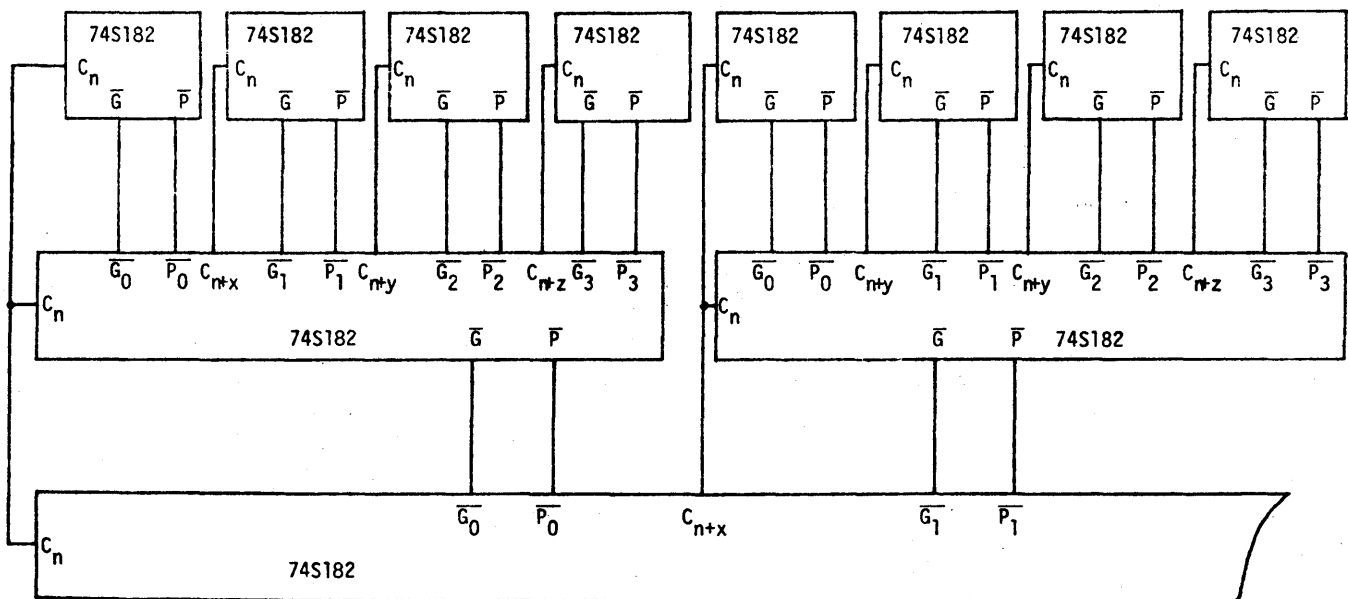


Figure 11—Interconnection of a carry-look-ahead generator.

port. A switch testing program (written in FORTRAN) exercises the module under test as follows: bit patterns are placed on the I/O port of the computer and the resulting signature from the module is compared with a table consisting of 'good' and 'faulty' signatures. The testing program steps through a table of patterns and returns a formatted report to the user.

## 7.0 CONCLUSIONS

The design and implementation of the banyan interconnection network for the Texas Reconfigurable Array Computer (TRAC) has been presented. We have shown that a switch designed within the restrictions of LSI (few pins, and  $>1000$  gates) can support a wide range of interconnection mechanisms that complement a powerful processor. The bidirectional switch is a powerful, unique, multifunctional network capable of setting up majority of partitions for SMIMD architectures. One of the most attractive characteristics of the switch includes: external control of the network at the top and the bottom levels only, cost-efficiency of  $O(n \log n)$ , carry look ahead and priority logic as well as time-multiplexed switching capability without degradation of the system performance. Modularity, expandability, LSI compatibility and fault tolerance potential make this network very desirable for support of varistructured, reconfigurable systems.

## 8.0 REFERENCES

1. Batcher, K. E., "The flip network in STARAN," *Proc. of the 1976 International Conference on Parallel Processing*, August 1976, pp. 65-71.
2. Feng, T., "Data manipulating functions in parallel processors and their implementations," *IEEE Trans. on Comput.*, Vol. C-23, March 1974, pp. 309-318.
3. Goke, L. R., "Connecting networks for partitioning polymorphic systems," Ph.D. Dissertation, Dept. E.E., U. of Florida, 1976.
4. Goke, L. R. and Lipovski, G. J., "Banyan networks for partitioning multiprocessor systems," *Proc. First Annual Symposium on Computer Architecture*, Dec. 1973, pp. 21-28.
5. Lawrie, D., "Access and alignment of data in an array processor," *IEEE TC*, Vol. C-24, No. 12, Dec. 1975.
6. Lawrie, D., "Memory-processor connection networks," University of Illinois, Report UTUCDCS-R-73-557, Feb. 1973.
7. Premkumar, U. V., Malek, M., and Lipovski, G. J., "A theoretical basis for switching structures," submitted to the 7th International Symposium on Computer Architecture, 1980.
8. Siegel, H. J., Mueller, P. T., Jr., and Smalley, H. E., Jr., "Control of a partitionable multimicroprocessor system," *Proc. 1978 ICPP*, 1978, pp. 9-17.
9. Siegel, H. J., "Analysis techniques for SIMD machine interconnection networks and the effects of processor address masks," *IEEE Trans. on Comput.*, Vol. C-26, Feb. 1977, pp. 153-161.
10. Siegel, H. J., McMillen, R. J., and Mueller, P. T., Jr., "A survey of interconnection methods for reconfigurable parallel processing systems," *Proc. of NCC*, 1979, pp. 529-542.
11. Stone, H., "Parallel processing with the perfect shuffle," *IEEE TC*, Vol. C-20, No. 2, Feb. 1971, pp. 153-161.
12. Thompson, C. D., "Generalized connection networks for parallel processor intercommunication," *IEEE Trans. on Comput.*, Vol. C-27, Dec. 1978.
13. Wu, C. and Feng, T., "Routing techniques for a class of multistage interconnection networks," *Proc. of the 1978 International Conference on Parallel Processing*, August 1979, pp. 197-205.
14. Pease, M. C., "The indirect binary n-cube microprocessor array," *IEEE Trans. on Comput.*, Vol. C-26, May 1977, pp. 458-473.
15. Lipovski, G. J. and Tripathi, A., "A reconfigurable varistructure array processor," *Proc. 1977 ICPP*, 1977, pp. 165-174.
16. Radoy, C. H. and Lipovski, G. J., "Switched multiple instruction, multiple data stream processing," *Proc. of Second Annual Symposium on Computer Architecture*, 1974, pp. 183-187.
17. Lipovski, G. J., "An organization for optical linkages between integrated circuits," *Proc. of NCC*, 1977, pp. 227-236.
18. Premkumar, U. V., Kapur, R., and Lipovski, G. J., "Interprocessor communication in TRAC," *Proc. of the first International Conference on Distributed Computing Systems*, 1979, pp. 51-62.
19. Seinowski, M. C. et al., "An overview of the Texas Reconfigurable Array Processor," submitted to *Proc. of NCC*, 1980.
20. Dujari, G. and Horne, P., "Node circuit for banyan switch of the Texas Reconfigurable Array Computer," Technical Report TRAC-9, University of Texas at Austin, January 1979.



# The advent of trusted\* computer operating systems

by STEPHEN T. WALKER

*Department of Defense*  
Washington, DC

## BACKGROUND

The need to trust a computer system processing sensitive information has existed since the earliest uses of computers. As the effectiveness of computer systems has improved, the desire to utilize them in increasingly more important and consequently more sensitive information processing applications has grown rapidly. Sensitive information must be protected from unauthorized access or modification. But without trusted internal access control mechanisms, the computer has to be treated as a device operating at a single sensitivity level.

Much has been learned about methods of assuring the integrity of information processed on computers since the emergence of operating systems in the early 1960s. Early efforts were primarily concerned with improvements in the effective use of the larger computer centers that were then being established. Information protection was not a major concern since these centers were operated as large isolated data banks. There were many significant hardware and software advances in support of the new operating system demands. Some of these changes were beneficial to the interests of information protection but since protection was not an essential goal at that time, the measures were not applied consistently and significant protection flaws existed in all commercial operating systems.

In the late 1960s, spurred by activities such as the Defense Science Board study (recently reprinted<sup>1</sup>), efforts were initiated to determine how vulnerable computer systems were to penetration. The "Tiger Team" system penetration efforts<sup>2</sup> record of success in penetrating all commercial systems attempted, led to the perception that the integrity of computer systems hardware and software could not be relied upon to protect information from disclosure to other users of the same computer system.

By the early 1970s we had long lists of the ways penetrators used to break into systems. Tools were developed to aid in the systematic detection of critical system flaws. Some were relatively simplistic, relying on the sophistication of the user to discover the flaw,<sup>3</sup> others organized the search into a set of generic conditions which when present often indicated an integrity flaw.<sup>4</sup> Automated algorithms were de-

veloped to search for these generic conditions, freeing the "penetrator" from tedious code searches and allowing the detailed analysis of specific potential flaws. These techniques have continued to be developed to considerable sophistication. In addition to their value in searching for flaws in existing software, these algorithms are useful as indicators of conditions to avoid in writing new software if one wishes to avoid the flaws which penetrators most often exploit.

These penetration aids are, however, of limited value in producing trusted software systems. For even if these techniques do not indicate the presence of any flaws it is not possible to prove a positive condition (that a system can be trusted) by the absence of negative indicators (known flaws). There will always be that one remaining flaw that has not yet been discovered.

In the early 1970s the Air Force/Electronics Systems Division (ESD) conducted in-depth analyses of the requirements for trusted systems.<sup>5</sup> The concepts which emerged from their efforts today are the basis for most major trusted computer system developments. The basic concept is a Reference Monitor or Security Kernel which mediates the access of all active system elements (people or programs) referred to as subjects, to all systems elements containing information (tapes, files, etc.) referred to as objects. All of the security relevant decision making functions within a conventional operating system are collected into a small primitive but complete operating system referred to as the Security Kernel. The three essential characteristics of this module are that it be: (1) complete (i.e., that all accesses of all subjects to all objects be checked by the kernel); (2) isolated (i.e., that the code that comprises the kernel be protected from modification or interference by any other software within the system); (3) correct (i.e., that it perform the function for which it was intended and no other function).

Figure 1 is a chronology of some of the major trusted computer system developments that have occurred since 1973. Following the ESD report in 1972, ESD and the MITRE Corporation began a series of efforts to implement security kernel based systems. The early efforts were limited to new operating systems built from scratch.<sup>6</sup> Also in 1973 a design study called the Provably Secure Operating System Study was initiated at SRI International.<sup>7</sup> From this effort emerged a system design specification process which is being widely used in later system developments and the preliminary design for a capability based architecture trusted system. From

\* A trusted computer operating system is one which employs sufficient hardware and software integrity measures to allow its use for simultaneously processing multiple levels of classified and/or sensitive information.

## COMPUTER SECURITY EVOLUTION

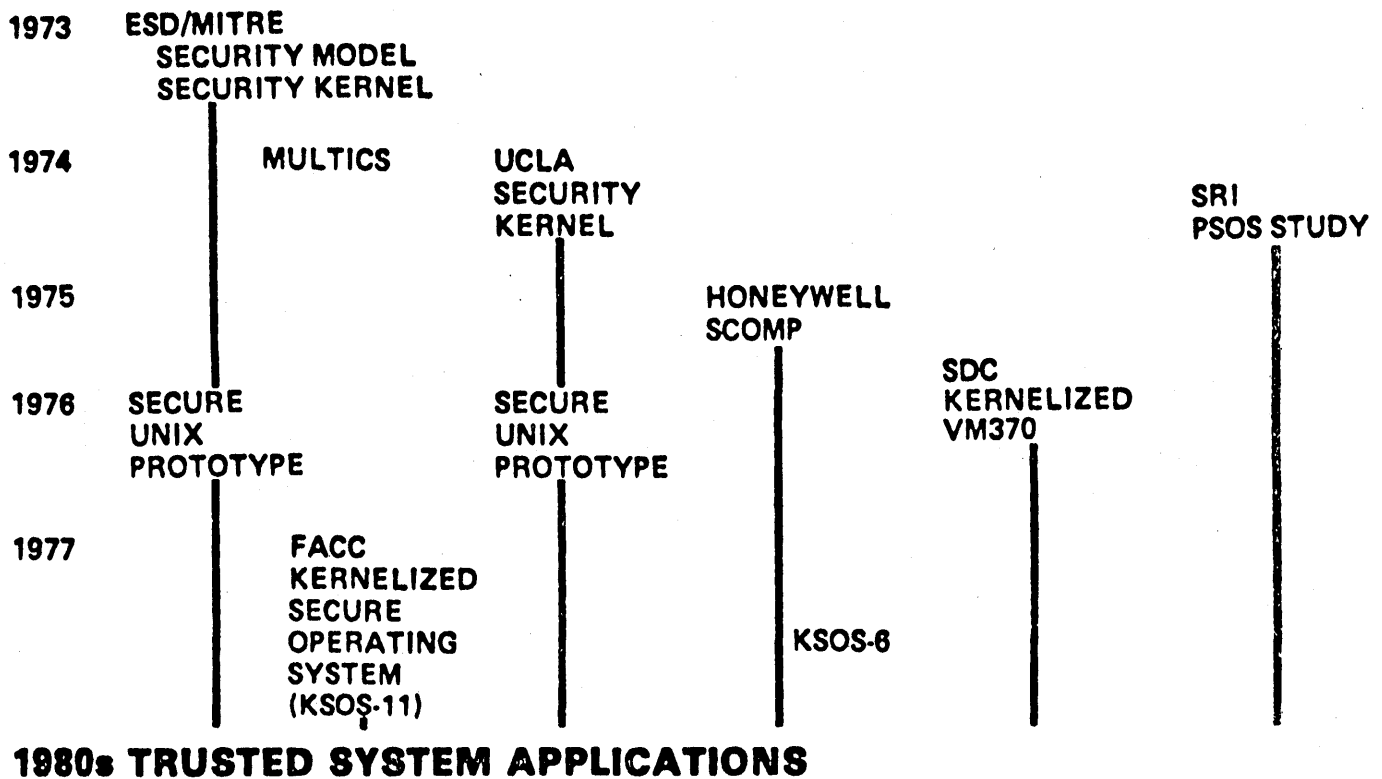


Figure 1.

1974 until 1976 much effort by MIT, Honeywell and MITRE was spent on designs of a security kernel base for MULTICS.<sup>8</sup> In 1974, work began at UCLA to develop a security kernel base for a virtual machine monitor for the DEC PDP-11 computer.<sup>9</sup> From this effort emerged a system design specification process which is being widely used in later system developments and the preliminary design for a capability based architecture trusted system.

In 1975 the Air Force initiated a research effort to design an improved hardware base for use as a Secure Communications Processor (SCOMP). Honeywell won the competitive procurement process with their Level 6 minicomputer. The hardware improvements to the Level 6, designed to improve the efficiency and effectiveness of a security kernel based software system, are called the Security Protection Module and are intended to be compatible options for the standard Level 6 computer.

In 1976 both the MITRE and UCLA projects to implement security kernel based systems began efforts to develop trusted prototypes of the UNIX (TM) operating system.<sup>10,11</sup> Also in 1976, the System Development Corporation (SDC) began the development of a kernelized version of the IBM VM370 operating system (KVM).<sup>12</sup> This system was demonstrated in a preliminary version in October 1979 and is expected to be available for specific DoD applications by late 1980.

In 1977 based on the success of the UCLA and MITRE trusted UNIX prototype developments, an effort was begun to develop a "protection quality" trusted system, entitled the DoD Kernelized Secure Operating System (KSOS) in a two-phase program. In the Design Phase, from August 1977 until April 1978, two competitively selected contractors (Ford Aerospace and Communications Corporation and TRW Inc.) developed detailed system designs. Following a careful evaluation of the two designs, the Implementation Phase contract was awarded to Ford Aerospace in May 1978.<sup>13,14</sup> This phase will implement, by Fall 1980, a production quality trusted operating system which is compatible with UNIX. This effort is sponsored by the Defense Advanced Research Projects Agency and several other DoD agencies, each of which has specific applications in mind for the system.

This KSOS implementation will be on the Digital Equipment Corporation PDP-11/70 computer in order to take maximum advantage of the widespread installed computer base and existing UNIX-compatible applications on that computer. However, the organization of this project has been substantially influenced by the possibility of implementations on hardware other than the PDP-11. The product of the Design Phase was a detailed system level specification.<sup>15</sup> This specification provides a functional description of each module of the security kernel and operating system. This

spec could be used to guide the implementation of versions of KSOS on other hardware architectures. The Honeywell Corporation has undertaken an internally funded KSOS development project for their SCOMP modified Level 6 minicomputer. Other implementations of the KSOS system are being studied by various organizations.

### TRUSTED OPERATING SYSTEM FUNDAMENTALS

An operating system is a specialized set of software which provides commonly needed functions for user developed application programs. All operating systems provide a well defined interface to application programs in the form of system calls and parameters. Figure 2 illustrates the relationship between the operating system and application software. The operating system interfaces to the hardware through the basic machine instruction set and to applications software through the system calls which constitute the entry points to the operating system. Applications programs (e.g., A, B and C) utilize these system calls to perform their specific tasks.

A trusted operating system patterned after an existing system is illustrated in Figure 3. The security kernel is a primitive operating system providing all essential security relevant functions including process creating and execution and mediation of primary interrupt and trap responses. Because of the need to prove that the security relevant aspects of the kernel perform correctly, great care is taken to keep the

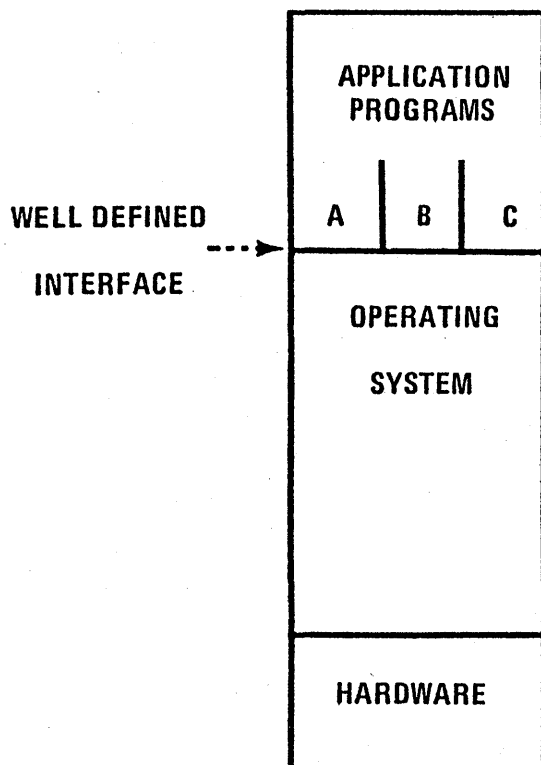


Figure 2.

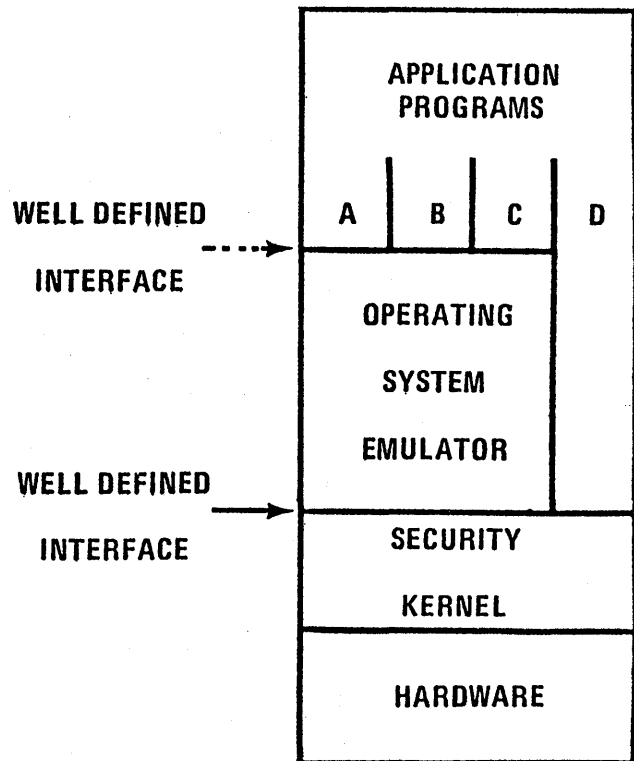


Figure 3.

kernel as small as possible. The kernel interface is a well defined set of calls and interrupt entry points. In order to map these kernel functions into a specific operating system environment, the operating system emulator provides the nonsecurity relevant software interface for user application programs which is compatible with the operating system interface in Figure 2. The level of compatibility determines what existing single security level application programs (e.g., A, B, C) can operate on the trusted system without change.

Dedicated systems often do not need or cannot afford the facilities or environment provided by a general purpose operating system, but they may still be required to provide internal protection. Because the security kernel interface is well defined and provides all the primitive functions needed to implement an operating system it can be called directly by specialized application programs which provide their own environment in a form tailored for efficient execution of the application program. Examples of this type of use are dedicated data base management and message handling systems.

Figure 4 illustrates the relationship between two typical computer systems connected by a network. Each system is composed of an operating system (depicted by the various support modules arrayed around the outside of each box) and application programs (e.g., A, Q, and R in the inner area of the boxes). The dotted path shows how a terminal user on System I might access File X on System II. Working through the terminal handler, the user must first communicate with an application program (A) which will initiate a network connection with the remote computer through the

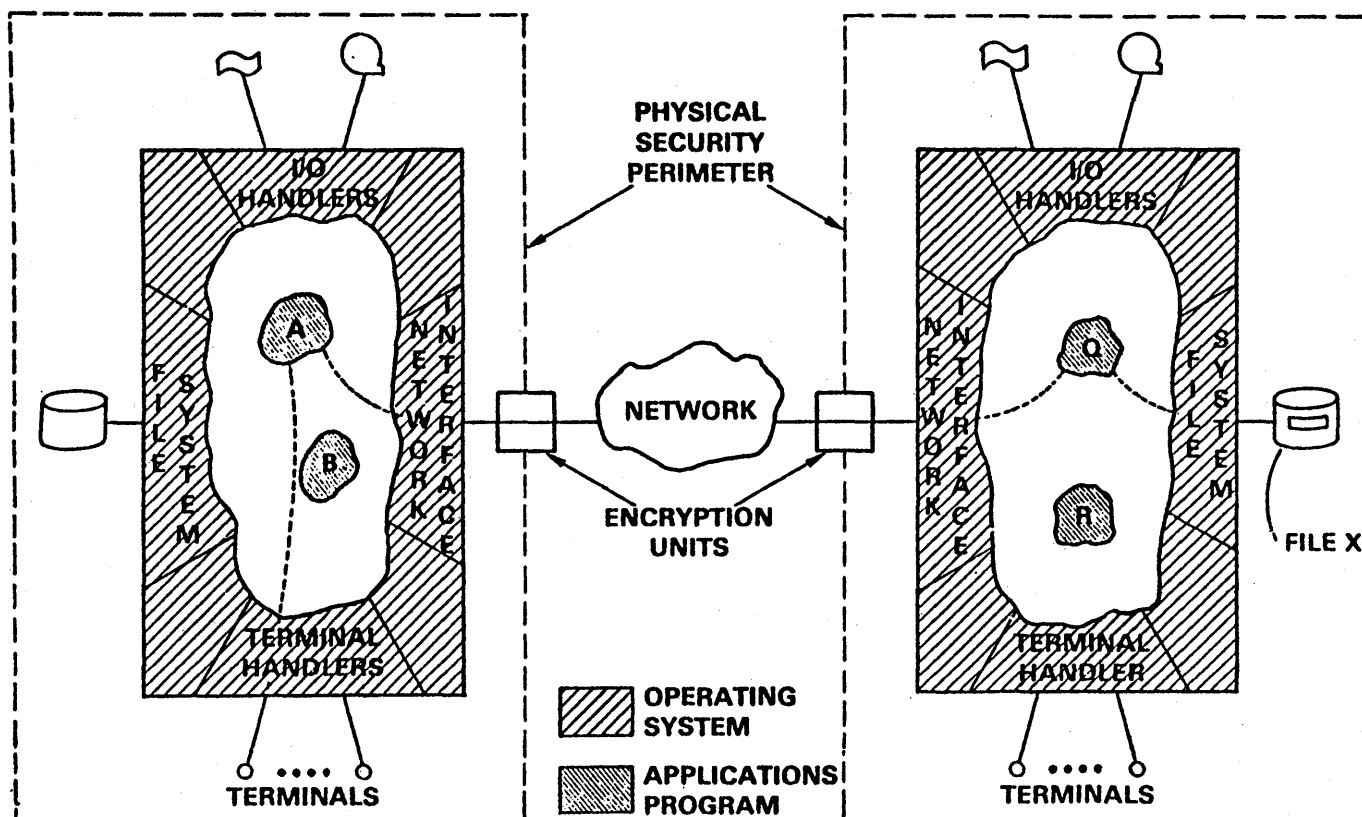


Figure 4.

network interface software. On System II an application program or a system utility (Q) is initiated on the user's behalf to access File X using the file system. Program Q could perform a data base update or retrieval for the user or it could arrange to transfer the file across the network to the local computer for processing.

When this scenario is applied in a secure environment, the two systems are placed in physically secure areas and, if the network is not secure, encryption devices are installed at the secure interface to the network as shown in Figure 4.

Figure 5 illustrates the function of the security kernel in the above scenario. Because the kernel runs directly on the hardware (Figure 3) and processes all interrupts, traps and other system actions, it is logically imposed between all "subjects" and "objects" on the system and can perform access checks on every event affecting the system. It should be noted that depending on the nature of the hardware architecture of the system, the representation of the kernel may have to include the various I/O device handlers. The DEC PDP-11, for example, requires that all device handlers be trusted and included in the kernel since I/O has direct access to memory. The Honeywell Level 6 with the Security Protection Module Option does not require trusted device drivers since I/O access to memory is treated the same way as all other memory accesses and can be controlled by the existing hardware mechanisms.

## SYSTEM SECURITY VULNERABILITIES

Protection is always provided in relative quantities. Guaranteed 100 per cent security is not possible with today's physical security measures, nor will it be with new computer security measures. There will always be something which can fail in any security system. The standard approach to achieving reliable security is to apply multiple measures in depth. Traditional locks and fences provide degrees of protection by delaying an intruder until some other protection mechanism such as a roving watchman can discover the attempted intrusion. With computer systems this "delay until detected" approach won't always work. Once an intruder knows about a security flaw in a computer system, he can generally exploit it quickly and repeatedly with minimal risk of detection.

Research on the security kernel approach to building trusted operating systems has produced a positive change in this situation. While absolute security cannot be achieved, the design process for trusted computer systems is such that one can examine the spectrum of remaining vulnerabilities and make reasonable judgments about the threats he expects to encounter and the impact that countermeasures will have on system performance.

A caution must be stated that the techniques described here do not diminish the need for physical and administrative

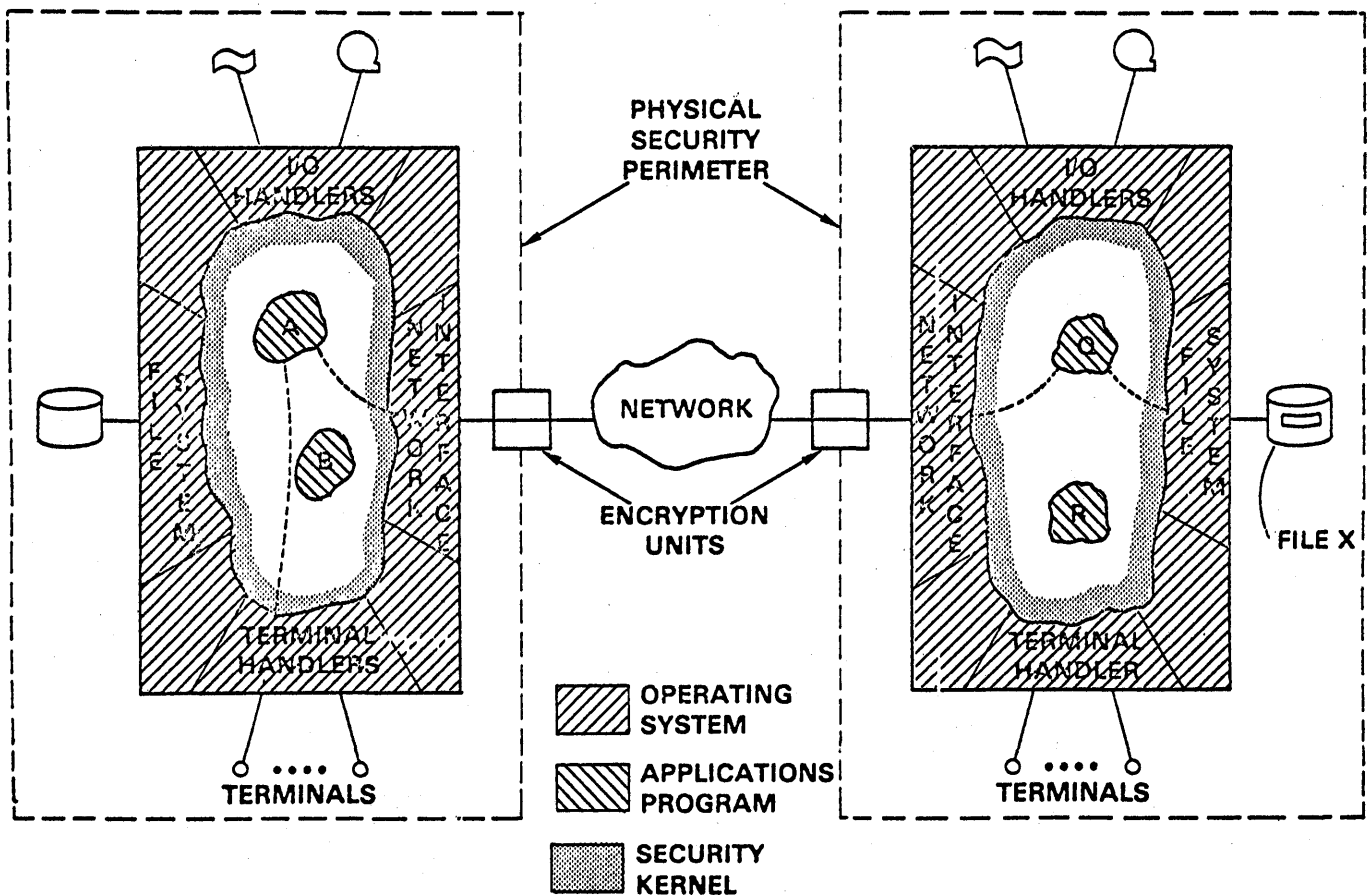


Figure 5.

security measures to protect a system from unauthorized external attack. The computer security/integrity measures described here allow authorized users with varying data access requirements to simultaneously utilize a computer facility. They provide this additional capability which relies upon the existing physical and administrative security measures rather than replacing them.

The nature of traditional physical and administrative security vulnerabilities encountered in the operation of computers with sensitive information is well understood. Only users cleared to the security level of the computer complex are allowed access to the system. With the advent of trusted computer systems allowing simultaneous use of computers by personnel with different security clearances and access requirements, an additional set of security vulnerabilities comes into play. Table I describes one view of this new vulnerability spectrum as a series of concerns. Each of these concerns was not serious in previous systems because there was no need or opportunity to rely on the integrity of the computer hardware and software.

The first category is the Security Policy which the system must enforce in order to assure that users access only authorized data. This policy consists of the rules which the computer will enforce governing the interactions between

system users. There are many different policies possible ranging from allowing no one access to anyone else's information to full access to all data on the system. The DoD security policy (Table II) consists of a lattice relationship in which there are classification levels, typically Unclassified through Top Secret, and compartments (or categories) which are often mutually exclusive groupings.<sup>16</sup> With this policy a partial ordering relationship is established in which users with higher personnel security clearance levels can have access to information at lower classification levels provided the users also have a "need to know" the information. The vulnerability concern associated with the security policy is assuming that the policy properly meets the total organizational security requirements.

The second general concern is the System Specification Level. Here the function of each module within the system and its interface to other modules is described in detail. Depending upon the exact approach employed, the system specification level may involve multiple abstract descriptions.<sup>17</sup> The vulnerability here is to be able to assure that each level of the specification enforces the policy previously established.

The next vulnerability concern is the high level language implementation. This category constitutes the actual module



TABLE I—Operating System Security Vulnerabilities

<u>Category</u>	<u>Function</u>	<u>Vulnerability Resolution</u>	<u>Relative Security Risk</u>
Security Policy	Establish security relationship between all system users, resources (e.g., DoD Security Policy)	Review	Moderate
System Specification	Establish policy relationship for each system module (e.g., Parnas I/O assertions)	For each module establish security assertions which govern activity	High
High Order Language Implementation	Transform System Specification Provisions for each module into (e.g. Fortran, PASCAL, C)	Manual or interactive validation that HOL obeys system spec	High
Machine Language Implementation	Transform HOL implementation into binary codes which are executed by hardware	Compiler Testing	Moderate
↑ Software (Installation Independent)			
-----			
↓ Hardware (Installation Dependent)			
Hardware Instruction Modules	Perform machine instructions (e.g., ADD instruction)	Testing, redundant checks of security relevant hardware.	Low -- except for security related hardware
Circuit Electronics	Perform basic logic functions which comprise instructions (e.g., AND, OR functions)	Maintenance Testing	Low
Device Physics	Perform basic electromagnetic functions which comprise basic logic function. (e.g. electron interaction)	Maintenance Testing	Very Low

implementation represented in a high order language (HOL) such as EUCLID<sup>18</sup> or PASCAL. This vulnerability involves the assurance that the code actually obeys the specifications. The next concern on the vulnerability list is the machine code implementation which includes the actual instructions to be run on the hardware. The step from HOL implementation to machine code is usually performed by a compiler and the concern is to assure that the compiler accurately transforms the HOL implementation into machine language.

The next level of concern is that the hardware modules implementing the basic instructions on the machine perform accurately the functions they represent. Does the ADD instruction perform an ADD operation correctly and nothing else? Finally, the last concerns include the circuit electronics and more fundamental device physics itself. Do these elements accurately perform in the expected manner?

As can be seen by analyzing this vulnerability spectrum, some of the areas of concern are more serious than others.

In particular, relatively little concern is given to circuit electronics and device physics since there is considerable confidence that these elements will perform as expected. There is a concern with hardware modules, though in general most nonsecurity relevant hardware failures do not pose a significant vulnerability to the security of the system and will be detected during normal operations of the machine. Those security relevant hardware functions can be subject to frequent software testing to insure (to a high degree) that they are functioning properly. The mapping between HOL and machine code implementation is a serious concern. The compiler could perform improper transformations which would violate the integrity of the system. This mapping can be checked in the future by verification of the compiler (presently beyond the state-of-the-art). Today we must rely on rigorous testing of the compiler.

The selection of the security policy which the system must support requires detailed analysis of the application require-

TABLE II—DoD Security Policy

## I. Non discretionary (i.e., levels established by national policy must be enforced)

	Compartments		
	A	B	C
Top Secret			
Secret			
Confidential			
Unclassified			

## Partially Ordered Relationship

Top Secret > Secret > Confidential > Unclassified

Compartments A, B, C are mutually exclusive

## Example:

User in Compartment B, level Secret can have access to all information at Secret and below (e.g., Confidential and Unclassified) in that compartment, but no access to information in Compartments A or C.

## II. Discretionary, "Need to know" - (i.e., levels established "informally").

ments but is not a particularly complex process and can be readily comprehended so the level of concern is not too high for this category.

The system specification and HOL implementation are the two areas which are of greatest concern both because of the complex nature of these processes and the direct negative impact that an error in either has on the integrity of the system. Considerable research has been done to perfect both the design specification process and methods for assuring its correct HOL implementation<sup>19,20,21,22,23</sup> Much of this research has involved the development of languages and methodologies for achieving a complete and correct implementation.<sup>24,25,26</sup>

As stated earlier this vulnerability spectrum constitutes a set of conditions in which the failure of any element may compromise the integrity of the entire system. In the high integrity systems being implemented today, the highest risk vulnerability areas are receiving the most attention. Consistent with the philosophy of having security measures in depth, it will be necessary to maintain strict physical and administrative security measures to protect against those lower risk vulnerabilities that cannot or have not yet been eliminated by trusted hardware/software measures. This will result in the continued need to have cleared operation and maintenance personnel and to periodically execute security checking programs to detect hardware failures. Over the next few years as we understand better how to handle the high risk vulnerabilities we will be able to concentrate more on the lower risk areas and consequently broaden the classes of applications in which these systems will be suitable.

## DIRECT VERSUS INDIRECT LEAKAGE PATHS

Computer system security vulnerabilities constitute paths for passing information to authorized users. These paths can be divided into two classes: direct (or overt) and indirect (or covert) channels.<sup>27,28</sup> Direct paths grant access to information through the direct request of a user. If an unauthorized user asks to read a file and is granted access to it, he has made use of a direct path. The folklore of computer security is filled with case histories of commercial operating systems being "tricked" into giving direct access to unauthorized data. Indirect or covert channels are those paths used to pass information between two user programs with different access rights by modulating some system resource such as a storage allocation. For example, a user program at one access level can manipulate his use of disk storage so that another user program at another level can be passed information through the number of unused disk pages.

Unauthorized direct access information paths can be completely eliminated by the security kernel approach since all objects are labeled with access information and the kernel checks them against the subject's access rights before each access is granted. The user who is interested only in eliminating unauthorized direct data access can achieve "complete" security using these techniques. Many environments in which all users are cleared and only a "need-to-know" requirement exists, can be satisfied by such a system.

Indirect data paths are more difficult to control. Some indirect channels can be easily eliminated, others can never be prevented. (The act of turning off the power to a system can always be used to pass information to users.) Some indirect channels have very high bandwidth (memory to memory speeds), many operate at relatively low bandwidth. Depending upon the sensitivity of the application, certain indirect channel bandwidths can be tolerated. In most cases external measures can be taken to eliminate the utility of an indirect channel to a potential penetrator.

The elimination of indirect data channels often affects the performance of a system. This situation requires that the customer carefully examine the nature of the threat he expects and that he eliminate only those indirect paths which pose a real problem in his application. In a recent analysis, one user determined that indirect path bandwidths of approximately teletype speed are acceptable while paths that operate at line printer speed are unacceptable. The assumption was that the low speed paths could be controlled by external physical measures. With these general requirements to guide the system designer it is possible to build a useful trusted system today.

## EARLY TRUSTED OPERATING SYSTEM APPLICATIONS

There are a number of classes of applications for which KSOS (either as a full UNIX compatible operating system or in the stand alone kernel mode) is well suited.<sup>29,30</sup> The first is an application called the Guard (Figure 6) in which two commercial untrusted data management systems, op-

## KSOS APPLICATIONS GUARD

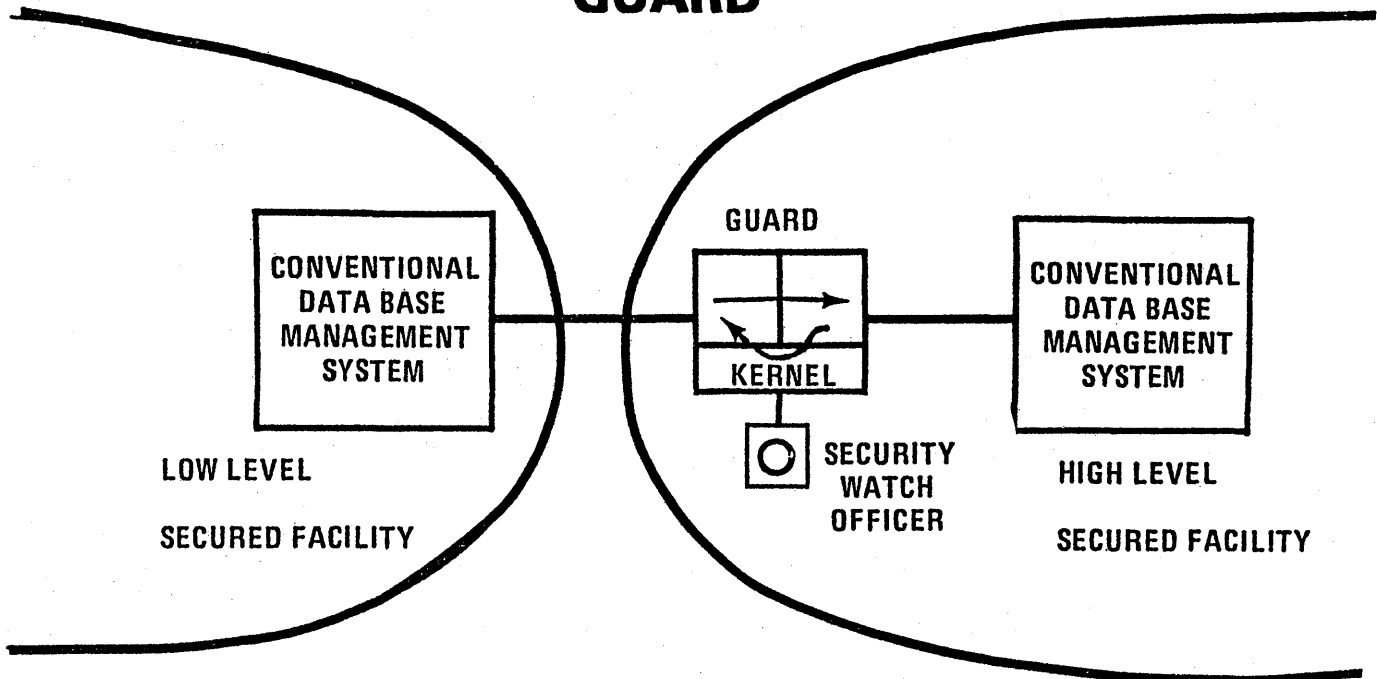


Figure 6.

erating at different security levels, are allowed to interact through a KSOS based security filter. Queries from the low level classified system are passed to the higher system through the Guard. Replies, which might contain information classified at the higher level, are sanitized by either operators or application programs on the Guard. Before information can be passed to the lower level system it must be presented by the security kernel to the Security Watch Officer for a determination of the appropriate classification of the sanitized reply. If the new classification is at or below the clearance level of the lower system, the reply can be forwarded. If not, it will be returned for further sanitization. This relatively simple application provides a very useful function and has wide utility in the DoD.

A second application class is that of trusted network front ends (Figure 7). In the interconnection of DoD computers by sophisticated data communications networks, extensive use is being made of front end minicomputers to offload many of the network protocol and terminal access functions from the mainframe systems. These networks must now be operated in a system high dedicated mode with all computers and terminals operating at the same security level. If these front end systems were implemented on security kernels, subnetworks of computers and terminals, each operating at its own security level, could be established. A set of co-operating trusted network front ends could provide a significant improvement to today's system high operating environment with no change required to the large systems.

A third general class of trusted system applications is that of message handling systems. The DoD has a wide range of requirements for systems of this type. One characteristic which crosses the entire spectrum of this application class and has thusfar not been satisfied is the need for internal integrity within the message system. In many cases security constraints preclude the handling of information from the full set of sources required by an organization because some element of that organization does not have the complete set of clearances required. The result is either a duplication of systems to handle different sources (with the resulting problems of stale or incomplete data) or the inaccessibility of information to sources that require it. If these message handling systems were built on a trusted base such as KSOS, they could make use of the access isolation mechanisms which it provides. Such a system could provide the integration of many information sources into a single environment with sufficient protection to isolate sensitive information.

These are only a few of the applications in the DoD (and in the private sector) that require operating systems with significant levels of internal integrity. Any system that processes sensitive information could benefit from the integrity provided by a trusted operating system. KSOS will not satisfy all the sensitive information handling needs that we now foresee, but as experience in applications like those described here yields a better understanding of more sophisticated applications, the systems which follow KSOS should be able to fulfill the growing number of requirements.

# KSOS APPLICATIONS

## TRUSTED NETWORK FRONT END

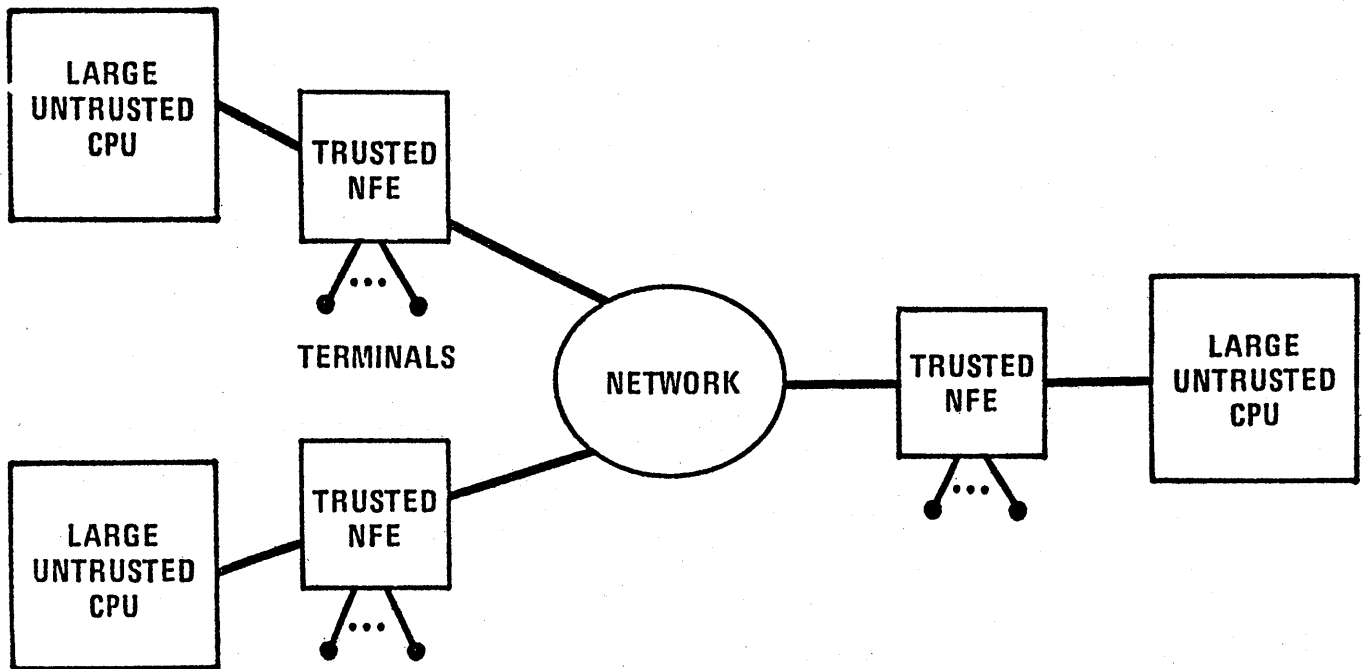


Figure 7.

### TRUSTED SYSTEM ACCEPTANCE

The terms "approval," "certification," "accreditation" and "validation" among others have frequently been used to describe some form of acceptance for use of a system in a particular security environment or security mode. Such environments include dedicated mode, periods processing (where the computer is operated at one security classification level for users with the same clearance and "need to know" for a period, stopped, cleared and then run at a different level for a period, stopped, cleared, etc.), system high (where all users are cleared to the same security classification level but may differ in their need to know) and simultaneous multiple security classification levels (referred to as multilevel secure or multilevel security mode where users with different personnel security clearance levels have simultaneous access to the same computer system).

In a multilevel secure environment, where the integrity of the computer hardware and software will be relied upon to the maximum to protect classified information, system approval has been particularly hard to contemplate. However, if a reasonable degree of integrity can be assured for the hardware and software security mechanisms then this combined with appropriate external security provisions should allow acceptance of trusted ADP systems.

The level of integrity afforded by the security kernel mech-

anism and the formal specification and verification process to which it is subjected, as applied in KSOS and KVM, should be sufficient for approval for use in a number of DoD applications in particular environments. The approval process will involve a detailed analysis of the risks to be encountered by the particular application/environment, the integrity measures inherent in the kernel based hardware and software, and the external physical and administrative measures which can be established.

It is important to understand that security is not a binary decision based only on the characteristics of the operating system or hardware. A particular system installed in one environment may be approved for use, while the same system in a different environment may be unacceptable. With an understanding of the spectrum of vulnerabilities that a trusted system will be subjected to and the external physical and administrative measures that are available, an evaluation of the threat posed by a particular application/environment can be performed to determine the suitability of a particular system in a particular environment. It is possible to establish general categories describing applications and environments and to evaluate systems such as KSOS and KVM to determine in which application/environments they should be suitable. The first applications of KSOS and KVM may be satisfactory for a limited set of environments. Later systems which are able to overcome more of the potential vul-

nerabilities should be acceptable in increasing broad application/environment combinations.

### DOD COMPUTER SECURITY INITIATIVE

This paper has thus far described the development and potential use of trusted systems such as the DoD's Kernelized Secure Operating System and Kernelized VM370 System. As significant as these developments are in themselves, there is a much more important "next step" that must be taken. First, some background is needed on the factors which have influenced the actions of the DoD and the computer industry.

The DoD relies on the computer industry to supply its general purpose computer hardware and operating system software. With the exception of limited special purpose systems, most computers in the DoD are commercial products utilizing vendor supplied operating systems. The cost of designing, implementing, and maintaining today's complex systems is so large and the underlying operating system support needs of the DoD are so little different from those of other ADP users that the expense of DoD unique operating systems cannot be justified.

The DoD and others have been for many years asking the vendors to build trusted operating systems. But since we were unable to clearly define what we meant by a trusted system, industry has been reluctant to undertake a serious development because of the high risk that when completed their product might be found unacceptable by either the DoD or other customers. As long as no one was able to demonstrate in detail what was being sought or what constitutes an acceptable product, there was little progress in the development of commercially available trusted systems.

One way to overcome this impasse is for someone (like the DoD) to build a trusted system, demonstrate that it is acceptable in real applications and provide detailed information on the techniques used in the development to the computer industry. If the technology used to build this system is suitable for application in general sensitive information handling environments, then there is a large and rapidly growing marketplace for such a product.

When viewed from this perspective the significance of KSOS and KVM takes on new dimensions. These efforts constitute the existence proof demonstration that a trusted operating system can be built and successfully used in DoD applications. Furthermore the approach used in building KSOS (i.e., the well documented detailed design phase including a top level formal specification of the kernel interface and a formal proof that it enforces an appropriate security policy, followed by an implementation phase) allows immediate transfer of this technology for early use in near term system developments. For the same reasons that the DoD cannot support its own operating system development efforts, it cannot fund multiple vendors to build such systems. But the demonstration of a technology suitable for widespread use in both government and industry should provide sufficient incentive to the computer industry to expend its own development resources to build a suitable line of trusted operating system products.

The DoD wishes to encourage the computer industry to develop, with their own resources, trusted operating systems with security provisions similar to those provided by the KSOS and KVM systems. In support of this, a DoD program is being planned to transfer information concerning the DoD's efforts to develop trusted operating systems and to evaluate industry developed systems which are submitted to the DoD for potential use in sensitive information handling applications.

Two seminars on the DoD Computer Security Initiative have been held in July 1979 and January 1980 at the National Bureau of Standards in Gaithersburg, Md.

### SUMMARY

This paper has described the background surrounding the development of several DoD trusted computer systems including the DoD Kernelized Secure Operating System and the Kernelized VM370 System, and the implications of these developments on the future uses of computers. These projects represent general purpose trusted operating systems intended for widespread use. They employ the concepts of a security kernel and an operating system emulator to provide maximum compatibility with existing software applications at a minimum investment of development cost and time.

KSOS and KVM are intended to demonstrate the trusted system development methodology and to provide a base for the useful application of trusted computer systems. They were not intended to be the ultimate answer to everyone's security problems but rather to point the way toward that goal. They also are part of an important initiative by the DoD to transfer an understanding of security kernel technology to industry to assist in the development of commercially available trusted systems.

The work described in this paper is the result of many years of research in trusted computer systems. The technology to build systems described here exists today. There is still much additional research required to develop trusted systems with the full flexibility which will be required in the future. We are recommending continued research into more sophisticated capabilities and we believe that the industry ties established in the DoD Computer Security Initiative will provide strong transfer mechanisms for future research accomplishments.

### REFERENCES

1. Ware, Willis H., "Security Controls for Computer Systems, Report of Defense Science Board Task Force on Computer Security," R-609-1, reissued October 1979, Rand Corporation, Santa Monica, CA.
2. Linde, Richard R., "Operating System Penetration," *Proceedings of the 1975 National Computer Conference*, 1975, pp 361-368.
3. Abbott, R. P., et al., "Security Analysis and Enhancements of Laboratory," Livermore, CA, National Bureau of Standards, Washington, DC, NBSIR 76-1041, April 1976.
4. Carlstedt, J., R. Bisbey and G. Popek, "Pattern Directed Protection Evaluation," USC Information Sciences Institute, ISI/75-31, January 1975.
5. Anderson, James P., "Computer Security Technology Planning Study,"

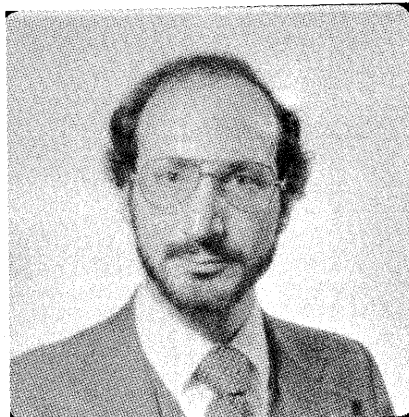
- James P. Anderson and Co., Fort Washington, PA, USAF Electronics Systems Division, Hanscom AFB, MA, ESD-TR-73-51, Vols I and II, October 1972 (AD 758206 and AD 772806).
6. Schiller, W. L., "The Design and Specification of a Security Kernel for the PDP-11/45," ESD-TR-75-69, The MITRE Corporation, Bedford, MA, May 1975 (AD A011712).
  7. Neumann, P. G., et al., "A Provably Secure Operating System: The System, Its Applications and Proofs," Final Report, Project 4332, SRI International, Menlo Park, CA, February 11, 1977.
  8. Schroeder, M., Clark, D., and Saltzer, J., "The MULTICS Kernel Design," *Proceedings of the Sixth Symposium on Operating Systems Principles*, West Lafayette, Indiana, November 1977.
  9. Popek, G. and Kline, C., "A Verifiable Protection System," *Proceedings of the International Conference on Reliable Software*, Los Angeles, CA, May 1975.
  10. Woodward, J. P. L. and Nibaldi, G. H., "A Kernel-Based Secure UNIX Design," MTR-3499, MITRE Corp, Bedford, MA, November 1977.
  11. Popek, Gerald J., et al., "UCLA Secure UNIX," *Proceedings of the 1979 National Computer Conference*, June 1979, pp. 355-364.
  12. Gold, B. D., et al., "A Security Retrofit of VM370," *Proceedings of the 1979 National Computer Conference*, June 1979, pp. 335-344.
  13. McCauley, E. J. and Drongowski, P. J., "A KSOS - The Design of a Secure Operating System," *Proceedings of the 1979 National Computer Conference*, June 1979, pp. 345-353.
  14. Berson, J. A. and Barksdale, Jr., G. L., "KSOS—Development Methodology for a Secure Operating System," *Proceedings of the 1979 National Computer Conference*, June 1979, pp. 365-371.
  15. Secure Minicomputer Operating System (KSOS), Computer Program Development Specification (Type B-5), Department of Defense Kernelized Secure Operating System, Ford Aerospace and Communications Corp., WDL-7932, September 1978.
  16. Bell, D. E. and LaPadula, L. J., "Secure Computer Systems: Mathematical Foundations and Model," M74-224, The MITRE Corp, Bedford, MA, October 1974.
  17. Robinson, L. and Levitt, K. N., "Proof Techniques for Hierarchically Structured Programs," *Communications of the ACM*, Vol. 20, No. 4, April 1977.
  18. Lampson B., et al., "Report on the Programming Language EUCLID," *SIGPLAN NOTICES*, Vol. 12, No. 2, February 1977.
  19. Popek, G. and Farber, D., "A Model for Verification of Data Security in Operating Systems," *Communications of the ACM*, September 1978.
  20. Millen, J., "Security Kernel Validation in Practice," *Communications of the ACM*, Vol. 19, No. 5, May 1976.
  21. Feiertag, R. J., et al., "Providing Multilevel Security of a System Design," *Proceedings ACM Sixth Symposium on Operating System Principles*, November 1977.
  22. Walker, B., Kemmerer, R., and Popek, G., "Specification and Verification of the UCLA UNIX Security Kernel," *Proceedings of ACM SIGOPS Conference*, December 1979, to be published in *Communications of the ACM*.
  23. Millen, J. K., "Operating System Security Verification," The MITRE Corp., M79-223, September 1979.
  24. Roubine, O. and Robinson, L., Special Reference Manual, SRI International, Menlo Park, CA, January 1977.
  25. Ambler, A., "Report on the Language GYPSY," University of Texas at Austin, ICSCA-CMP1, August 1976.
  26. Holt, R. C., et al., "The EUCLID Language: A Progress Report," *Proceedings of ACM 78 Conference*.
  27. Lampson, B., "A Note on the Confinement Problem," *Communications of the ACM*, Vol. 6, No. 10, October 1973.
  28. Lipner, S., "A Comment on the Confinement Problem," Fifth Symposium on Operating System Principles, Austin, TX, November 1975.
  29. Woodward, J. P. L., "Applications of Multilevel Secure Operating Systems," *Proceedings of the 1979 National Computer Conference*, June 1979, pp 319-328.
  30. Padlipsky, M. A., Biba, K. J., and Neely, R. B., "KSOS-Computer Network Applications," *Proceedings of the 1979 National Computer Conference*, June 1979, pp 373-381.



## Software Management

Much has been written about establishing an acceptable engineering discipline within organizations that develop and maintain software. Papers proliferate on how different engineering methods can be applied and about other tools available to help apply them. Yet, the technical discipline established using these methods cannot work unless a solid management foundation is created upon which they can operate.

The purpose of this double session is to expose the audience to the project management techniques that are being used today by successful managers to realize an acceptable product on time and within budget. This panel session is not about software management tools and techniques. Rather, it stresses practical ways of implementing a management discipline employing modern tools and techniques sometimes against strong opposition. Panel members represent organizations that are implementing such techniques and that understand their ramifications.



Donald Reifer  
*Area Director*

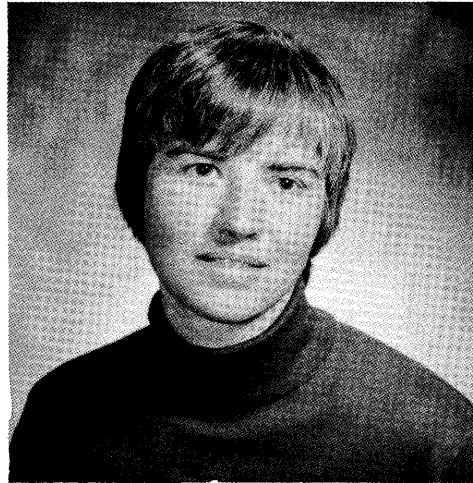




## **Software Engineering Technology Transfer**

Software that has been developed with the tools, techniques, and methodologies of software engineering has proven to be more reliable, efficient, and maintainable than conventionally developed software. The use of these software engineering techniques by data processing practitioners would result in higher quality software and an attendant increase in effectiveness of an organization's data processing function.

This session will address the problems and solutions in transferring this software engineering technology to the practitioners in the field. The first paper will deal with defining the needs of these practitioners, the second paper will present the methodologies available, and the third paper will provide the concepts of an integrated facility for developing software. The panelists will present their experiences and ideas from the viewpoint of the government, technical societies, universities, and industry.



Lorraine Duvall  
*Area Director*



# An integrated support software network using NSW technology

by RICHARD A. ROBINSON and EMILY A. KRZYSIK

*Rome Air Development Center  
Griffiss AFB, New York*

## INTRODUCTION

The problems with support software and the management structure employed in acquiring major weapon systems within the Air Force have been documented in an excellent fashion by General McCarthy,<sup>1</sup> and his remarks are used to introduce the subject of this paper. In addition, plans are currently under way to tie together the many organizations involved in major system acquisitions,<sup>2</sup> and to utilize ARPANET and National Software Works (NSW) technology<sup>3,4</sup> as the framework for a series of technology demonstration(s). Finally, the reader should note that substantial efforts are currently under way within the Air Force to standardize and control programming languages and compilers.<sup>5,6</sup>

"Program managers are faced with a major challenge brought about by the rapid expansion in use of digital computers in our modern weapon systems. It is estimated that the Air Force spends in excess of one billion dollars annually to make required changes to existing embedded computer programs. This cost will continue to grow as we bring into the inventory an increasing number of more complex digital systems. Given potential snowballing support costs, there is the real possibility that the Air Force will not be able to afford the required support posture for future weapon system embedded computers unless we change our current management philosophy.

It appears that major software systems and their attendant management structure have grown without an overall master plan or long range goal to guide the developer, maintainer, and user organizations. Today there are at least 37 different regulations and policy letters on Air Force management of software systems. Some of these regulations and policies provide conflicting guidance.

In the late 1960s and early 1970s, a typical weapon system had less than 100,000 words of embedded software, usually handling only one or two fairly straightforward functions. Current estimates place the operational software for the Joint Tactical Information Distribution System (JTIDS) in excess of 400,000 words. The operational software for the E-3A is over 500,000 words implementing 275,000 instructions. This represents only 9 percent of the total E-3A weapon system software.

This volume of embedded software is a major factor in the

high support costs. Another is the expense of changing mature software. A representative cost to develop a line of embedded software would be \$40-\$75, depending on complexity of the program and development tools available. Once the weapon system has been fielded and the configuration baseline has been established, it may cost upwards of \$4000 to change that same line of software.

With software's inherent flexibility we can add functions or integrate systems to provide rapid and effective response to threat changes or to advances in technology. We stand to lose this inherent flexibility and seriously impair our mission readiness posture unless we develop a strong support capability whether it be organic or contractor furnished."<sup>1</sup>

## WHAT IS SUPPORT SOFTWARE?

For purposes of this paper, support software will be defined "as the set of programs you need to develop the software you want."<sup>7</sup> A representative set of programming aids (or tools) is as follows:

- Communications Aids
- Compilers
- Linkers
- Mgmt Info Systems
- Online Doc Aids
- Standards Enforcers
- V & V Aids
- Requirements Analysis Tools
- Assemblers
- Online Editors
- Data Base Mgmt Systems
- Code Auditors
- Automated Testing Systems
- Simulators

## HOW IS SUPPORT SOFTWARE ACQUIRED?

In order to appreciate why support software is so costly, it might be best to review in more detail how support software gets acquired on a major system acquisition. Typically,

a user organization (USER) identifies an operational need. A subsequent study effort is awarded to further describe the operational requirements and/or develop functional specifications. A System Project Office (SPO) is created for coordinating system acquisition efforts. Procurement actions are initiated, a prime contractor (DEVELOPER) is selected, components of the job are identified, digital systems are integrated, etc. Note that little if any attention is paid to support software. The basic support software is found or developed by the prime contractor, and the criteria used is that "if something is once proved adequate, it will be used again because of schedule, monetary, and other constraints."<sup>7</sup>

The support software that gets developed in this fashion is developed out of desperation rather than by design. It is generally crude and inefficient because it has not been developed by specialists, nor has it been developed for use by others. Neither does it take advantage of current technology. Definite tradeoffs occur between the sophistication of the support software, and "getting on" with the job. For example, a program manager may decide it is not cost effective to use an on-line editor on his project because it is currently not available, it is buggy, or his staff has no experience with it. Therefore he uses punched cards, etc. As a result, the support software ends up being very expensive, inefficient, and unusable during subsequent phases of the system life cycle, or on other programs.

Once the system has been developed and tested, it is transitioned to the logistics command (MAINTAINER) for support on behalf of the operational command (USER). The accompanying support software is frequently viewed as inadequate or unusable because there is no documentation, they are not familiar with it, they have not been trained to properly support it, and frequently are not able to take over and maintain the system using it. As a result, the logistics command is forced into assuming development contractor responsibilities rather than concentrate on maintenance (or resource management) functions.

### WHY DO WE HAVE PROBLEMS WITH SUPPORT SOFTWARE?

In view of the above remarks, let's try and summarize why we have problems with support software so that we can get a better handle on what to do about it. According to Softech's analysis of the problem:<sup>7</sup> (1) it is expensive to develop and funds may not be available; (2) it takes time to develop and time may not be available; (3) it is not planned that others may want to use it; (4) it is not developed nor maintained by specialists; and (5) user access (via networks) has not been provided nor is it conveniently available.

Even if it is available, support software is generally not suitable during subsequent phases, or on other programs because: (1) it is developed as a "one-time" application specific package; (2) it is operating system and machine dependent; (3) it is crude and inefficient, and is poorly documented; and (4) it requires such extensive modification that re-development may be easier, more timely and less expensive.

### WHAT CAN WE DO ABOUT IT?

It is proposed that the following steps be taken to address the support software problem:

- Establish an Integrated Support Software Network — to provide DMU personnel with convenient and timely access to a repository of proven, high quality tools.
- Provide a staff of software specialists and consultants — to assist DMU personnel in selecting tools from the repository, and in assessing the utility of the selected tools.
- Standardize, control, and distribute the tools that have been determined to be most useful and cost effective.
- Maintain the tools as required by Air Force (or DoD) policy, and keep abreast of new technology developments.
- Demonstrate feasibility and practicality of the ISSN concept.

With the support of ISSN staff personnel, DMU personnel would log directly into the ISSN and request an inventory list of available tools. They would then select from that list those of most interest, and try those they find to be most useful for their needs. Once the decision is made to procure the tools, they would be delivered via the network to their respective machines. Supporting documentation would be provided on-line, and in hard copy form. If this were not feasible because of machine non-compatibility, or possibly machine non-availability, DMU personnel would simply use the requested tool kit or remotely located ISSN machine(s) until a dedicated, project-owned machine became available. The tool kit, or selected tools would then be transferred to their host environment for dedicated project use.

### HOW DO WE DO IT?

As illustrated in Figure 1, the overall ISSN concept can be broken down into the following elements: (1) a computer network; (2) the NSW system; (3) a "core" facility; (4) a number of general purpose/experimental machines; (5) a number of project-specific (native) machines; and (6) a professional staff of software specialists, dedicated to supporting the needs of DMU personnel, including maintenance, training and documentation.

#### *The network*

A computer network, e.g. the ARPANET, would be used to tie together the major machines involved in the acquisition of a major weapons system. It would also be used to provide real-time communications between the many organizations involved in formulating requirements and specifications, and during design, development, and subsequent maintenance of the system. The network is designed to provide efficient communications between heterogeneous computers so that hardware, software, and data resources can be conveniently

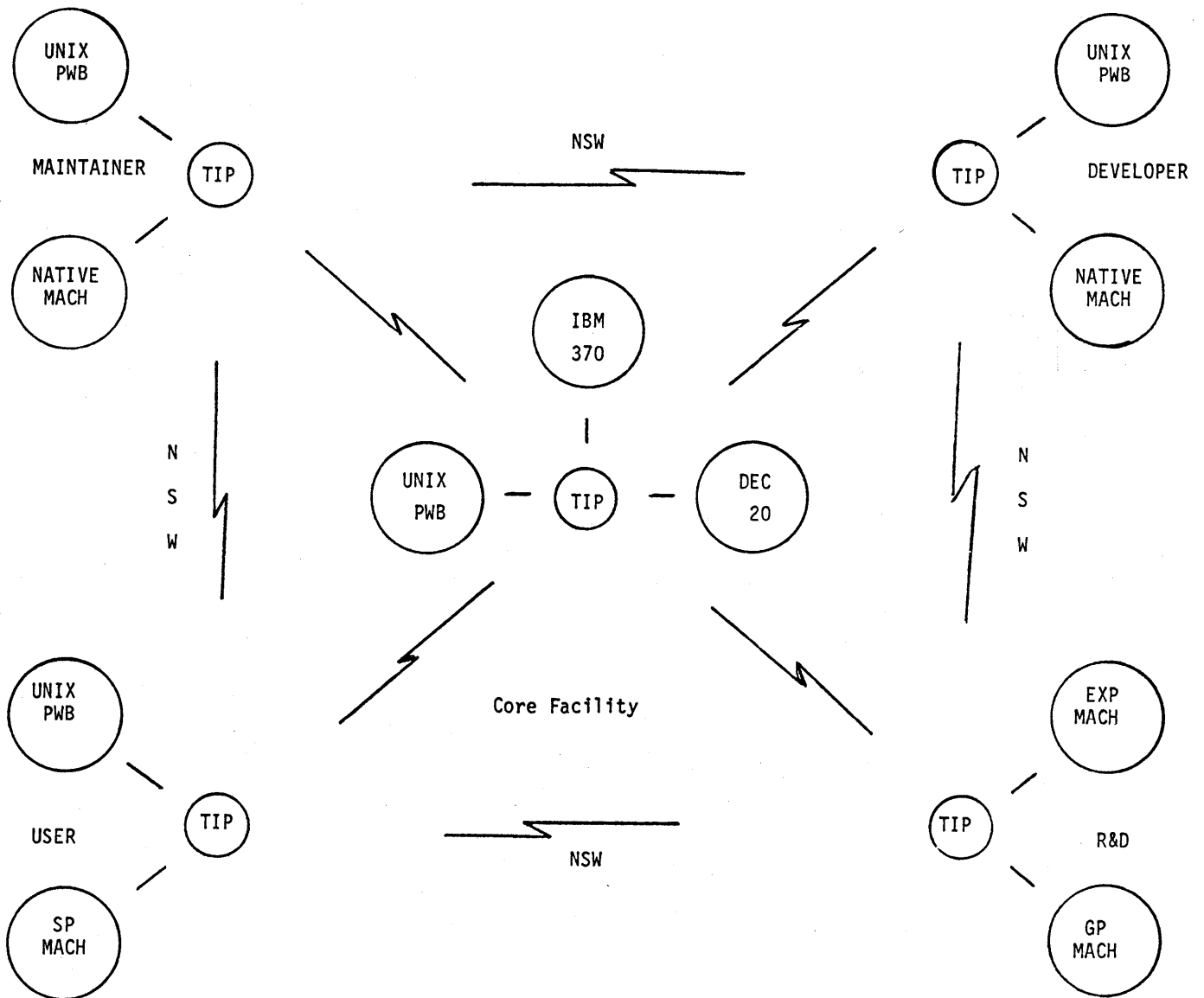


Figure 1-A network-based ISSF.

and economically shared by a wide community of users. The ARPANET currently links a wide variety of computers at Defense Advanced Research Projects Agency (DARPA) sponsored research centers and other DoD and non-DoD activities in CONUS, Hawaii, Norway, and England.

*The NSW system*

Once a network connection had been established, the National Software Works (NSW) system would be used to provide DMU personnel with convenient access to the distributed computer resources, including those available within the core facility (described below). The NSW is a distributed software system which resides upon the ARPANET host machines, and provides a user with single point access to

resources (or tools) on those machines. It also obviates the need for DMU personnel to know host operating system or file system details.

*The core facility*

The major element within the ISSN is the "core" facility or tool repository, where DMU personnel can get the latest information on tool technology, and where they can assess the suitability of and possibly acquire high quality tools for use on their respective projects. The core facility might consist of three types of machines - DEC 20 (TOPS 20), IBM 360 (OS), and DEC 11/70 (UNIX/PWB), all suitably hosted on the network described above. Extensions can readily be made to include other mainframe computers, e.g. UNIVAC

1110 (EXEC 8). Geographic location of the machines is not important because of the network connection. It is important, however, that the core facility be professionally staffed and run by a responsible organization.

The tool repository contains a variety of tools (or toolkit). The fundamental notion is that the same toolkit would be available to DMU personnel throughout the system life cycle. Individual tools would be simply added to or removed from the toolkit from time to time, depending upon DMU needs. A basic toolkit might consist of an on-line editor (TECO), a compiler (JOVIAL), a program support library (PSL), and an automated testing tool (JAVS). Additional support software would be obtained from the tool repository, as required. The decision on which tools to include in the toolkit would depend on the particular acquisition phase, and the functions to be performed.

The core facility can also be thought of as a repository where prospective tool vendors can install proprietary or unproven tools so they can be properly evaluated by core facility/DMU personnel. Of particular interest is the issue of whether the tools properly communicate with each other. The "do it yourself" syndrome invariably leads to the development of stand alone tools that do not communicate with each other, e.g. the many dialects of JOVIAL that currently exist, and of the many problems that occur because of the prevalence of divergent interfaces. More rapid assessment and evaluation of this aspect of tool technology is possible through the use of NSW technology. When DMU personnel become convinced of the applicability of a tool, or a tool kit, they will import the selected tool(s) (with supporting documentation) onto their "native" machines.

#### *General purpose/experimental machines*

These are machines that currently exist within the NSW System as tool bearing hosts (TBH) and would be utilized by DMU personnel for such things as electronic mail, training, off-loading when dedicated (native) resources are saturated or are not available, or when specific tools (or tool kits) are not available on the native machine(s). Machines available for this purpose are TENEX (DEC-10) and TOPS-20 (DEC-20) machines at USC-ISI, an IBM 3033 at UCLA, a TOPS-20 (DEC-20) at RADAC, a MULTICS (H6180) at RADAC, an EXEC 8 (UNIVAC 1110) (location to be determined), and a UNIX (DEC 11/70) system. Note that the "core" machines identified above are also on this list. Actual details will depend upon DMU needs.

It should be noted that general purpose/experimental machines can be used for supporting specific phases of the acquisition process. For example, dedicated SREM or CAD-SAT machines can be used to support requirements definition, a PDL machine for design support, etc. It is not unrealistic to suggest that a machine be totally dedicated to a specific function, e.g. requirements analysis, and would service multiple DMU organizations. Access to these resources would be obtained (and controlled) at each DMU site.

#### *Project-specific (or native) machines*

These are usually owned and operated by contractor personnel or government agencies, and are located on-site and under the control of contractor or government personnel. For the ISSN concept to be demonstrable, it is necessary that the operating system on the native machine be compatible with one of the core facility machines. Exceptions to this rule should generally be discouraged or forbidden because of the obvious and frequently detrimental impact on associated support software. Specific tools (or toolkit) to support development of the weapon system can then be imported from the core facility using the network connection. These tools will consist mostly of standardized, high quality tools that have been registered with the Federal Software Exchange and are maintained and serviced by a professional staff of software specialists. If native or dedicated program resources are not available, or are otherwise saturated during peak load conditions, the developer will also have the option of using a remotely located general purpose/experimental machine.

It is required that the native machine(s) be on or have convenient access to the ARPANET, and be machine/operating system compatible with the core facility machine/operating system, i.e. a TOPS-20, OS 370, UNIX PWB. This is necessary primarily because of the cost (and time) of re-hosting the support software obtained from the core facility; also because of the requirement for core facility staff to maintain the rehosted tools. Upon completion of development and testing, the DEVELOPER turns over project owned machines to the logistics command (within the Air Force) and they are installed in a support center specifically set up to maintain each major weapon system. These facilities are located within the individual air logistic centers (ALCs). In order to complete the scenario, it is necessary that these support centers also be on or have convenient access to the ARPANET to ensure continued use of the same tools used during earlier phases.

#### *A professional staff of software specialists*

For the ISSN concept to be demonstrably successful, the staff located within the "core" facility must provide timely service to DMU personnel, and must effectively maintain and control the tools which reside within the repository. This includes the servicing of software trouble reports (STR) submitted by DMU personnel on tool problems.

#### RECENT EVENTS ARE BEGINNING TO LOOK PROMISING!

Although recent efforts in language standardization and control look promising, much remains to be done to change the basic procurement/management practices that are used to acquire and support major systems. Some of the changes that are currently under way that support the ISSN concept

are as follows:

#### *MIL-STD-1589A*

- A single language for Command & Control
- Standard interface is likely
- LCF standardization & control possible

#### *MIL-STD-1750*

- One instruction set
- Standard linker possible
- Standard assembler possible
- Standard debugger(s) possible

#### *MIL-STD-1553B*

- Standard I/O protocols likely
- Environmental models are easier to adapt
- Aids V&V tool standardization

#### *DAIS Tech Demo*

- Flexible avionics executive
- Flexible structured digital systems

#### *AFLC Tech Demo*

- Establish network/NSW applications
- Install network connections
- Install standard support facilities (PWB)
- Provide NSW resources

#### WHAT CAN BE ACCOMPLISHED?

Now that we have described the elements of an integrated support software network, what can be expected to happen if it were suitably staffed and demonstrated? It should be possible to demonstrate:

- The support software development burden can be removed from the system program offices (SPO). Support software requirements, specifications and high quality tools can be provided to the SPO's, and contractor and SPO personnel can conveniently assess the quality of the support software before buying or committing it!
- The support software maintenance and enhancement burden can be transitioned to the logistics command, where the distribution and use of tools employed during system acquisition can be used to responsibly maintain major weapon systems.
- The developer, maintainer and user efforts can be focused on improving techniques for advancing weapon

system technology rather than building up an inadequate support software capability on each program and for each phase.

- The quality of weapon systems can be improved.
- Life cycle costs can be substantially reduced.
- Support software can be made immediately available for subsequent programs.
- Most importantly, it will be easier to establish a corporate memory from phase to phase (within a project) and across projects, and effectively apply new advances in tool technology.

#### WHAT IMPROVEMENTS CAN BE EXPECTED?

Some of the improvements that can be expected if proven, high quality support software is utilized by DMU personnel are:

##### *Requirements analysis*

- Life Cycle Costing Possible
- Less Costly Maintenance
- Distribution Problems Resolved
- Training Problems Simplified
- More Timely Development
- Responsible R&D Support

##### *Documentation*

- Standardized and Controlled
- Distribution On-Line
- Under Configuration Control

##### *Integrated tool kit*

- Library of Tools (selectable)
- High Quality Tools
- Reliability Demonstrated
- Effectiveness Demonstrated
- Interoperability Demonstrated

##### *Availability (of ISSN)*

- For Training Support
- For Tool Assessment (planning)
- For Tool Evaluation (project)
- For Tool Utilization
- For On-line Assistance

##### *Adaptability (of ISSN)*

- For Special Problems
- Readily Available
- Maintenance & Engineering



*Support*

- Adaptation Assistance
- Requirements Analysis Assistance
- For Local Site Support

**SUMMARY**

This paper has described an integrated support software network (ISSN) which can be used by the DEVELOPER, MAINTAINER and USER organizations in acquiring and maintaining large, computer-based systems. It focuses on the problems of technology transfer (it is virtually non-existent) within the software engineering business and how an "integrated" support software network could be used to alleviate or solve many of these problems. Networking and NSW technology are proposed as the vehicle for tying together the various machines, resources and organizations involved in major system acquisitions. Application of this technology provides a means for changing acquisition man-

agement practices within the Air Force, and should result in substantial cost savings and more timely delivery of major weapon systems.

**BIBLIOGRAPHY**

- 1 McCarthy, General James, Brig Gen, U.S. Air Force, AFALD/AQ letter, entitled "Embedded Computer Software Management Concept."
- 2 "NSW Technology Demonstration Plan for Air Force Logistics Applications," dated 18 October 1979.
- 3 ARPANET Information Brochure, Code 535, Defense Communications Agency, Washington, D.C. 20305, 1978. (Available from NTIS as AD A052672).
- 4 "Semi Annual Technical Report on NSW for the period 12 December 1978 - 30 June 1979," by Massachusetts Computer Associates, Inc., July 1979 CADD-7907-1201.
- 5 "Higher Order Language Control Facility," Air Force Contract F30602-79-C-0032.
- 6 "Requirements for ADA Programming Support Environments," "STONE-MAN" document, Office of the Under Secretary of Defense, February 1980.
- 7 SOFTECH seminar presentation entitled, "Solving JOVIAL Support Problems for Embedded Computers; A Controlled Approach."

# The role of an information analysis center in software engineering technology transfer\*

by JON MARTENS and LORRAINE DUVALL

*IIT Research Institute*  
Rome, New York

## THE PREDICAMENT

As software engineering advances into its second decade, the ideas, principles and practices conceived in its first decade need to be assimilated into a workable set of tools and techniques that can be dispersed to software developers for their use in the production of software.<sup>1</sup> The need for this technology transfer is clear and immediate. Without the proper transfer of software engineering technology from software engineering researcher to developer, the software world will be unable to extricate itself from its present predicament. This predicament has been characterized by Meyers as follows:

"The general character of the software predicament can be seen clearly, although consistent numbers with which to characterize it more precisely are hard to come by. Because less expensive hardware is bringing more applications within economic reach, the amount of software to be developed is increasing. Also because more software is already in existence, there is more to be maintained. But the productivity of programmers is improving rather slowly, especially by the standards of hardware price/performance, with the result that the overall cost of software development is tending to increase."<sup>2</sup>

The predicament presents a rather ominous picture to be sure. Essentially, the problems of today in the software world are more difficult, but the solutions to the problems do not seem to be effective. The result is a losing battle if things continue as they have. One reason for the losing battle may be that the software world is trying to solve today's problems with yesterday's solution techniques. For example, several recent surveys have shown that the transfer of software engineering is at a standstill and that people are still developing software as they were five years ago.<sup>3</sup>

## A POSSIBLE SOLUTION

"I firmly believe that technology transfer is the primary means we have to combat the software problems the industry has been experiencing."<sup>4</sup>

\* This research was supported by Contract No. F3060-78-0255, from the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441.

Reifer's statement points a way to the beginning of a solution to the software predicament that exists today. Technology transfer needs to bring new and effective solutions from the software engineering researcher to the software developer. Unfortunately, the process is not quite as easy as it sounds. Setting up direct communication links between researcher and developer will not insure that technology is transferred over these links. The wholesale importation of the latest software engineering techniques, in fact, just invites disaster. The developer needs to understand the techniques so they can be evaluated within the context of the development environment. This is essential. Another way of stating this is by citing Reifer's Technology Risk Principle.

"Technology should only be used when the risk associated with it is acceptable."<sup>5</sup>

Some of the technologies may indeed be acceptable in terms of risk; however, the developer needs to understand the technology and all of its myriad applications before any consideration can be made to its transfer to the development environment. This technology transfer business is more than giving lectures or writing journal articles about the latest technologies. It involves a certain amount of information synthesis and analysis so that the ultimate receiver can evaluate the technology's worth. If no benefit is perceived by the developer, no transfer of technology is going to occur. Of that we can be certain.

## SOME SOLUTION MECHANISMS

Although the technology transfer process is a difficult one, there are mechanisms in place today within the software engineering community to effect the transfer of technology. Wasserman, for example, cites four major mechanisms (and their attendant shortcomings).<sup>6</sup>

1. University graduates go to work in software development settings.

*Problem:* New graduates generally have positions of low visibility and responsibility and have not received any

development experience within the university environment.

2. University faculty serve as consultants to industry.  
*Problem:* Consultants often opt for more "interesting" (i.e. more research-oriented) situations as they come about and have little continuity within one company. They also lack the same software development expertise that their students lack.
3. Industry people go to the university.  
*Problem:* Sabbaticals are not open to many people and, besides, they are rarely taken by people with major project responsibilities.
4. Industry people attend short professional development courses.  
*Problem:* The direct application of the techniques learned in these courses is often difficult to transfer to a typical work situation.

These four techniques all involve some level of inter-personal dealings in an environment different than one the person is used to. Because the performance of individuals in such a situation has such a wide variance, it is difficult to obtain a consistent level of technology transfer. The crucial process of transferring the context of ideas generated in a researcher environment to a developer environment is a difficult achievement when attacked on a one-to-one basis. Nevertheless, these techniques should be effective if the "right" person for the job can be identified.

Studies in scientific and technical information dissemination have identified such a person.<sup>7</sup> Called a "gatekeeper," this person has the important technology transfer ability to interface the outside world (i.e. the research community) to the inner realm of the development organization. Much of the time, however, these people tend to be senior technical staff and not high level managers who could influence the company to adapt a new software engineering technology.

The professional societies and the journal literature serve as an aid to technology transfer in a more formal manner than the inter-personal techniques. For example, many of the societies arrange and conduct tutorials to disseminate information about software engineering technologies. Along the same lines, the journals of the professional societies publish articles about the latest techniques. Although these techniques are formal, their effectiveness is limited by the very structure of the professional groups and journals.

Professional groups, by their nature, are essentially special interest groups created to advance the interest and ideas of a relatively narrow area of interest. For this reason, the societies and their journals are more for the purpose of intra-group rather than inter-group communication. Both researcher and developer have their own societies, and both sides recognize the need and importance of the technology transfer issue. But the very structure of any society is not designed to facilitate a process such as technology transfer. The required interface mechanism between developer and researcher are not strong for the societies or the journal literature.

## INFORMATION ANALYSIS CENTER AS TECHNOLOGY TRANSFER AGENT

A formal mechanism that can perform the interfacing role between researcher and developer for the purpose of technology transfer in the software world does exist. The mechanism is the information analysis center. Specifically, the information analysis center for software engineering is named the Data and Analysis Center for Software, hereafter referred to as DACS.

The purpose and objective of an information analysis center transcend those of a technical information center or library. The technical information center and library provide bibliographic services as their main commodity. Information analysis centers, as their name may imply, provide information analysis and synthesis services as their primary commodity. It is these analysis and synthesis services that serve as the core of the technology transfer mechanism within the information analysis center. As has been previously discussed, a precondition for the transfer of software engineering technology from the researcher to the software developer is a thorough understanding of the technology and its attendant implications in terms of risks and benefits within the developer's environment. It is this essential precondition to the technology transfer process that can be provided by the synthesis and analysis of information within the information analysis center. Although the previously discussed mechanisms are all useful to some extent, they do not possess the match of capability and task that exists between the information analysis center and the technology transfer task. In software engineering, the requisite synthesis and analysis skill are provided by the DACS as the software engineering information analysis center.

## THE INFORMATION ANALYSIS CENTER CONCEPT

The formal concept of the information analysis center was expressed in 1963 by the nuclear physicist Alvin Weinberg.<sup>8</sup> Weinberg had prepared a report at this time which was to serve as somewhat of a landmark document in the field of national scientific and technical information policy. The information analysis center concept did not originate in the report; in fact, the report listed over 400 organizations in the nation it considered as meeting the criteria of an information analysis center.<sup>9</sup> Highlighting the contribution information analysis centers could have in managing the nation's technical information was one of the major concerns of the report. An excerpt from the report explains the worth of the information analysis center concept.

"The activities of the most successful (information analysis) centers are an intrinsic part of science and technology. The centers not only disseminate and retrieve information, they create new information. . . . In short, knowledgeable scientific interpreters who can collect relevant data, review a field and distill information in a manner that goes to the heart of a technical situation are more help to the overburdened specialist than a mere pile of relevant documents."<sup>10</sup>

Although this excerpt does not explicitly mention technology transfer, that process of "... distilling information in a manner that goes to the heart of a technical situation" is certainly central to the process of technology transfer. The receiver of the technology must be able to understand and evaluate the technology on his own terms. The analysis and synthesis of information about a technology by the information analysis center matches the need and is the driving force behind the center's ability to act as an effective technology transfer agent. The fit between technology transfer and the information analysis center is a natural one; the statement Weinberg made about the value of the synthesis and analysis process is as valid and crucial today in the area of software engineering as it was in 1963 when the statement was made. By synthesizing and analyzing software engineering information into a form that is comprehensible and relevant to the software developer, the DACS has an important role as technology transfer agent in the field of software engineering. This task of synthesizing information in software engineering is more difficult than it might be with other fields because of the breadth and dynamic state of software engineering at the present time. But these very characteristics of software engineering may result in bigger pay-offs when the technologies are transferred than may be the case with a narrower or more static discipline. The task is harder, but the potential benefits resulting from software engineering technology transfer may be greater.

#### A TWO-WAY STREET

Technology transfer is generally thought of as a one-way street from researcher to developer. Simply, technology, which originates with the researcher, is transferred to the software developer who is the ultimate user of the technology. In its broadest sense, however, technology transfer is a two-way street. It is a two-way street because the researcher needs the experience of the developer for guidance in future research efforts. Both good and bad experiences with technology provide the researcher with valuable information to improve the technology.

Information analysis centers are good places for the developer to transfer software experience information. Just as the information analysis center interfaces between the transfer of technology from researcher to developer, it can provide the interface for the flow of experience data from developer to researcher. The synthesis and analysis skills of the information analysis center are just as useful and relevant for both directions of information flow. A large part of the DACS effort is expended in this regard, particularly in the organization of software experience data into databases that can be used for the evaluation and analysis of software engineering technologies.

#### DACS—A SOFTWARE ENGINEERING INFORMATION ANALYSIS CENTER

In June of 1975, the Rome Air Development Center (RADC) contracted with IIT Research Institute (IITRI) to

design a center that would acquire, analyze, and disseminate information on software engineering technology. The Air Force recognized the need for such a center to serve the government, industrial, and university community as a focal point for software development and experience data.

DoD has traditionally recognized the worth of information analysis centers. Several organizations within DoD sponsor a number of centers. For example, nine DoD Information Analysis Centers are managed and funded by the Defense Logistics Agency (DLA). In keeping with the nature of all information analysis centers, the centers are responsible for the acquisition, analysis, evaluation, and dissemination of scientific and technical information to the managers, scientists, engineers, and technicians they support. Among the specialized areas the centers deal with are electronic hardware reliability, metals and ceramics, and machinability. As part of its responsibilities, each center serves as an information and technology transfer agent within its own area of technical expertise.

A contract to establish the DACS was awarded to IIT Research Institute (IITRI) by RADC in August 1978. When fully implemented and operational, the DACS will provide a centralized source for current, readily-usable data and information concerning software technology. This software information resource will: (1) aid the program manager in planning and monitoring software projects; (2) supply experience data to software research projects; (3) provide baselines for software development methods comparisons; (4) foster the use of uniform terminology; (5) aid in establishing data collection guidelines and standards; and (6) distill and disseminate information on software projects.

The benefits expected to be accrued by members of the software engineering community, both developers and researchers, are: (1) valuable savings of scientific and engineering manhours in locating data and information; (2) rapid application of the latest technologies via technology transfer; (3) elimination or minimization of duplication of effort; (4) reduction of software costs and improved performance; (5) minimization of program delays and schedule stretchouts.

All of the objectives of the DACS are related to the process of technology transfer. Through the first objective, the DACS will aid the technology transfer process by supplying information about software technologies to the developer in an understandable format. The second objective deals with the feedback from developer to researcher, as discussed earlier, that is required for refinement of technologies. Objective Number 3 will facilitate technology transfer by providing a basis for the evaluation of technologies in developer-related terms. Objectives 4 and 5 are objectives that will streamline the technology transfer mechanism. Uniform terminology (objective 4) and data collection guidelines (objective 5) are essential to aid communication between researchers and developers. Objective 6 is, in itself, the essence of the information analysis center technology transfer process—the ability to synthesize and distill information about the technology.

The benefits of the DACS, when they are realized, will result in efficient and effective technology transfer (benefits

1-3) and a start at escaping from the software predicament which the Air Force, the Department of Defense, and the entire software community are facing (benefits 4 and 5). Although these benefits are ambitious, the framework of the information analysis center provides an excellent mechanism for achieving these goals.

#### HOW DACS OPERATES

Objectives cannot be fulfilled or benefits realized unless a methodology to achieve these ends is established and followed. Information analysis centers serve technology transfer by synthesizing, distilling, analyzing and repackaging information. The DACS, as an information analysis center technique, provides a mechanism for synthesizing and analyzing the information concerning software engineering technology. Since the DACS is currently a pilot facility, the techniques are just being formatted, utilized, and tested. Although these techniques are still in a test mode, they should be adequate as they follow the techniques used for other technical IAC's with a fair amount of success. A functional model of the techniques is shown in Figure 1.<sup>11,12</sup>

#### DACS ACTIVITIES AND PRODUCTS

Two major components make up the technique represented in Figure 1: (1) building an information base about software engineering technology and, (2) transferring information

about the technology in a form that can readily be understood, evaluated, and used to the advantage of the software developer by the processes of information analysis and synthesis. Each of the major components consists of several sequential processing steps with each processing step resulting in the output of a particular information type. As a pilot facility, the DACS has more experience with the earlier steps, but all steps have been utilized to some extent.

Building the technology information base has been a major effort and concern of the DACS to date. This process consists of (1) information collection and, (2) information organization. From a technology transfer viewpoint, the purpose of these two steps is to prepare the information base so that it can later be synthesized into a form more suitable for technology transfer.

Information collection is being actively pursued by the DACS. The professional society publications are reviewed, as are conference proceedings: reports on new research projects, trade journals, and technical reports from universities, government, and industry. Information on new and previous software engineering research is the primary result of this review and collection procedure. Software experience data is also being collected by the DACS. At the present time, seven major datasets have been assembled and two are currently being assembled. Collection of all this information is, of course, necessary prior to any synthesis or analysis of the data that is required for technology transfer.

Information organization is the next step in the process of building a technology information base. Information man-

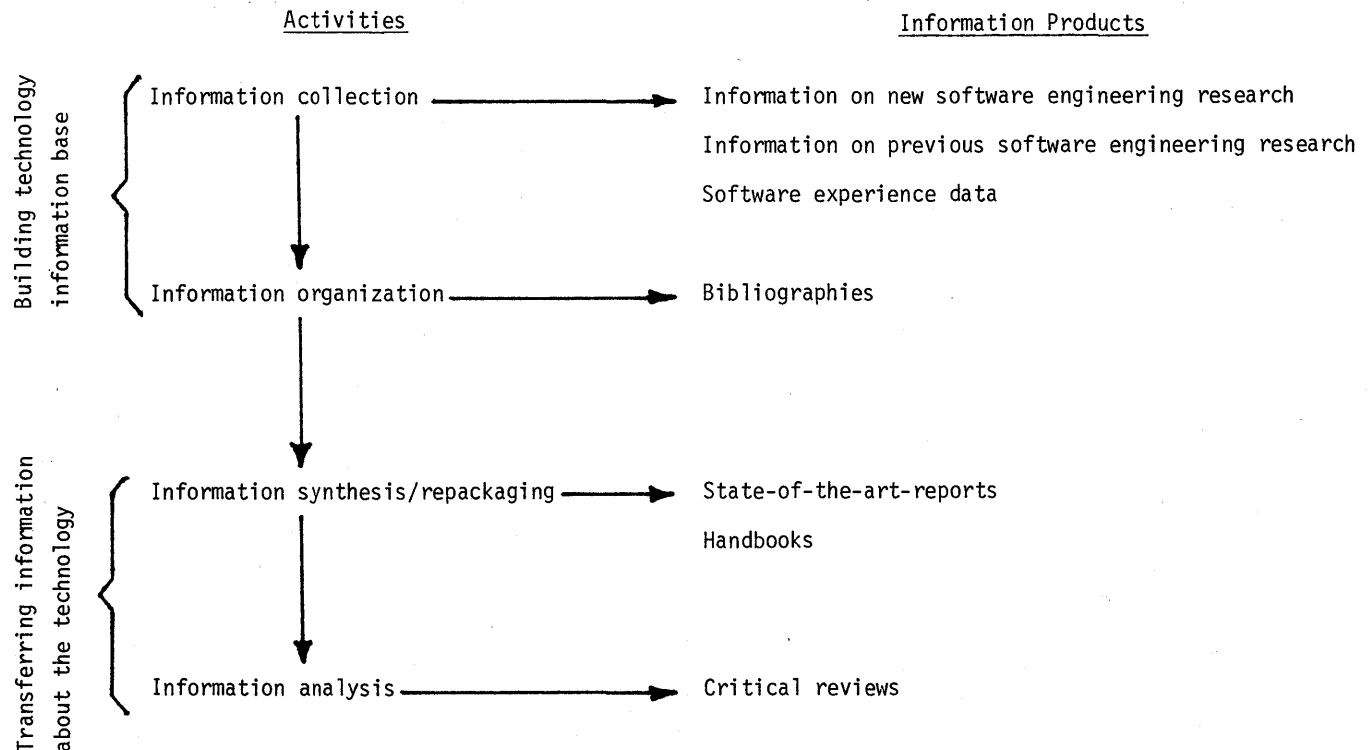


Figure 1—DACS activities and information products.

agement skills, such as indexing, abstracting, and information storage and retrieval, are used in this phase. The information collected in the first step is indexed, abstracted, and entered into a computerized information retrieval system. Custom bibliographies can then be produced by the retrieval system by specifying subject keywords or by qualifications on other fields such as author or title. These bibliographies are of use to the research community and developers in locating results of research. At this point in the process, the information is organized as final preparation for the steps of information synthesis and analysis that are so important to the information analysis center's contribution to technology transfer. These first two steps (information collection and organization) are performed by most technical libraries. The next two steps, however, separate the information analysis center from the technical library.

Information synthesis, the next step, is the core of the technology transfer process. Information generated by the researcher must be digested and made palatable for the developer. If at all possible, the information must be presented in a format to allow cost-benefit assessment by the developer. Depending on the novelty and age of the technology, this may not always be possible. At the very least, however, the information must be distilled so the basic concepts and principles are explained in a language that the developer can comprehend. This is not an easy task. Considerable skills in communications and expertise in the technology are required to produce the handbooks and state-of-the-art reports that are the outputs of this technology transfer process.

DACS has produced a state-of-the-art report on quantitative software models.<sup>13</sup> Research in this area has been extensive and has the potential for being used by developers, so this subject area was a prime candidate for this initial effort. With these thoughts in mind, the state-of-the-art report was produced with two major features to facilitate technology transfer. The first feature was a description of the salient characteristics of the model, such as data parameters, key equations and relationships, and experiences in using the model. This feature enables the developer to understand the model's concepts and capabilities.

Synthesis of information was carried one step further to produce the second feature of the report. A matrix was prepared to correlate model with data parameters. By using this matrix, a developer can quickly determine what models could be used with the data parameters available to the developer. If data parameters were unavailable, the cost of collecting them could be weighed against the benefits of the model as presented in the description.

Additionally, DACS has published a glossary of software engineering terms.<sup>14</sup> This glossary should aid the technology transfer process by providing a reference point for uniform terminology.

One last step is included in the technology transfer process: information analysis. Analysis goes one step beyond synthesis by providing an evaluation of the technology. As the center makes the transfer from pilot to full-scale operation, this analysis effort will be pursued. One target of analysis that DACS would like to examine is the effective-

ness of modern programming practices. At the present time, data sources for such an effort are being collected and organized.

## USER PERCEPTIONS

A recent survey mailed with the *DACS Newsletter* is currently being analyzed. One interesting result in the light of the role of the DACS as technology transfer agent, is the interest of the respondents in state-of-the-art reports. These reports, as previously mentioned, are one of the primary products of the DACS.

Questionnaire respondents were given a list of the seven types of information processed and/or generated by an information analysis center and were asked the following:

"For your job, please rate the value of the following types of information (1 = most valuable)"

Table I summarizes the results. Although these results are preliminary, the results point to a definite preference for surveys of current technologies. The final results of the survey will be used to plan future DACS services and products.

## CONCLUSION

The need for technology transfer in software engineering is clear and essential if the software world is to escape from

TABLE I—User Survey: Preliminary Results

Profile of user:	
User	Percent of Respondees
Researchers:	28%
Developers:	70% (Managers 38%, Programmer/analyst 32%)
Not designated:	3%
	100%
Information value:	
Percent of respondents ranking an information type as 1, 2, or 3 on a scale of 1 (most valuable) to 7 (least valuable).	
Information Type	Percent
Information on new software research	57%
State-of-the-art reports	75%
Bibliographies	15%
Software experience data	24%
Information on previous software research	35%
Handbook information	37%
Critical reviews	37%

its present predicament. Various methodologies and mechanisms exist for this transfer process. One process, using an information analysis center as a transfer agent, is appealing because the capability of the information analysis center as synthesizer of technology fits the task of transferring technology in such a way that the software developer and user can understand and evaluate the technology. As a software engineering information analysis center, the DACS is fulfilling the role of transfer agent. Mechanisms are in place to synthesize the software engineering technology and disseminate it to software developers. During its operation, DACS has built an information base of software engineering and used that base to synthesize the technology into several handbooks and state-of-the-art reports. Response to the reports has been encouraging, and the DACS plans to issue more in its role as a software engineering technology transfer agent.

## REFERENCES

1. Wasserman, Anthony and Belady, L. A., with contributions from Susan L. Gerhart, Edward F. Miller, Jr., William Waite and William A. Wulf, "Software Engineering: The Turning Point," *Computer*, September 1980, pp. 30-41.
2. Meyers, Ware, "The Need for Software Engineering," *Computer*, February, 1978, pp. 12-26.
3. McClure, Robert M., "Software—The Next Five Years," *Digest of Papers Comcon Spring 76*, p. 607, as cited in Ware Myers, "The Need for Software Engineering," *Computer*, February 1978, pp. 12-26.
4. Reifer, Donald, "The Nature of Software Management: A Primer," in Donald Reifer, *Tutorial: Software Management*, IEEE Computer Society, March 1979, p. 5.
5. —, "The Nature of Software Management: A Primer," in Donald Reifer, *Tutorial: Software Management*, IEEE Computer Society, March 1979, p. 5.
6. Wasserman, Anthony and Belady, L. A., with contributions from Susan L. Gerhart, Edward F. Miller, Jr., William Wait and William A. Wulf, "Software Engineering: The Turning Point," *Computer*, September 1978, pp. 30-41.
7. Paisley, W., "Information Needs and Uses," in C. A. Caunda, *Annual Review of Information Science and Technology*, Vol. 3, Encyclopaedia Britannica, Chicago, as cited in Lauren B. Doyle, *Information Retrieval and Processing*, Melville Publishing Company, Los Angeles, 1975, p. 390.
8. Weinberg, Alvin, "Science, Government, and Information," a Report of the President's Science Advisory Committee, Panel on Science Information, January 1963, as cited in Lorraine Duvall, *Software Data Repository Study*, RADC-TR-76-387, December 1976, p. 1-1.
9. —, "Science, Government, and Information," a Report of the President's Science Advisory Committee, Panel on Science Information, January 1963, as cited in Lorraine Duvall, *Software Data Repository Study*, RADC-TR-76-387, December 1976, pp. 1-2.
10. Weinberg, Alvin, "Science, Government, and Information," a Report of the President's Science Advisory Committee, Panel on Science Information, January 1963, as cited in Alan Rees, "Functional Integration of Technical Libraries, Information Analysis Centers," in Alan Rees, *Contemporary Problems in Technical Library and Information Center Management: A State-of-the-Art*, American Society for Information Science, 1974, p. 119.
11. Brady, Edward L., "The Role of Information Center in Engineering Information System," in *Proceedings of the National Engineering Conference*, Engineer's Joint Council, 1964, as cited in Allan Rees, "Functional Integration of Technical Libraries, Information Centers and Information Analysis Center," in Alan Rees, *Contemporary Problems in Technical Library and Information Center Management: A State-of-the-Art*, American Society for Information Science, 1974, p. 119.
12. Bishop, Ethelyn and Nisenoff, Norman, *An Application of Market Research Techniques to the Dissemination of Scientific and Technical Information*, NSF/DS11751-13211 A01, November 1977.
13. *Software Engineering Research Review, Quantitative Software Models*, (SRR-1), March 1979.
14. *The DACS Glossary, A Bibliography of Software Engineering Terms*, October 1979.

# Considerations in the transfer of software engineering technology

by MICHAEL J. MCGILL

*Syracuse University*  
Syracuse, New York

## INTRODUCTION

The development of new and increasingly efficient techniques of software engineering seems to be impressive to everyone except the professional software developers. In a recent summary of a panel discussing software engineering problems to be faced during the 1980's, Wasserman notes:

Not surprisingly, the panel concluded that the problems of the '80's look very much like the problems of the '70's and depressingly similar to the problems of the '60's. The basic questions. . . were presented by the chairman as follows:

- 1) How can well engineered products and systematic procedures for their creation be developed?
- 2) How can the body of software engineering techniques be applied to existing systems?
- 3) How can technology be transferred more effectively from the research community to software developers?<sup>1</sup>

This paper will examine the prospects for the use of software engineering techniques in the near future by the "average programmer." The characteristics of technology transfer will be examined and put into perspective with software engineering.

## THE PROGRAMMING ENVIRONMENT

Programming is an intellectually challenging exercise with at least two factors, size and complexity, identified as significant by Gries.<sup>2</sup> Size is important because any one person can remember only a small portion of a programming system. However, if size was the only consideration, then the task of programming, like that of knitting, might be challenging but not intellectually stimulating. Complexity factors combined with size factors insure that programming will remain an intellectual challenge. Complexity may be either associated directly with the computational process or with the determination of the correctness of the program.

Structured programming and design are approaches to the size and the complexity problems and have a significant impact on the computer industry. The concept of GOTO—less code has led to case studies of systems with tremendous improvements in productivity, reliability, and reduced main-

tenance costs. The IBM system developed for the New York Times was *the* example used by many proponents of structured techniques. Unfortunately, one discovers that numerous maintenance problems persisted even in this exemplary system.

Modular design and top-down design continue to be catch phrases for programmers. Many techniques have been defined for the definition, description, implementation, and testing of programs. The techniques allow for hierarchical descriptions, structured charts, decision tables, control graphs, etc. Each is designed to enhance the programmer's effectiveness and efficiency. However, Yourdan and Constantine note that, "to say that the average programmer's design process is organized, or structured, would be charitable."<sup>3</sup>

The approaches to the design, development, and evaluation of a software product are continually increasing in number and sophistication. For instance, Gries<sup>4</sup> points out three approaches to programs:

### *Enumerative reasoning*

Used to understand the sequence of statements, conditional statements and some uses of GOTO. It is an approach to understanding and showing the correctness of execution paths.

### *Mathematical induction*

Used to understand iteration and recursion. Induction may be used to show that procedures are correct irrespective of the number of times the procedure is invoked.

### *Abstraction*

Used to isolate the relevant properties or qualities of an object so that one can focus on what the object does. In this manner one examines data types, variables, etc., to insure correct operation.

The ideal is to have a program that can be proven correct. Recognizing these approaches is, however, quite different



from applying them. Most programmers are aware of software engineering devices and in fact recognize the need for increasingly effective methods. Procedures for proving the correctness of a program have been considered for more than a decade.<sup>5</sup> The procedures are the results of efforts by numerous computer scientists. Scientists, even computer scientists, tend not to be professional programmers. Professional programmers are more properly categorized as very high level technologists. Therefore, the transfer of software engineering technology is really an issue which concerns the transfer of information about available tools, techniques, and procedures from the computer scientist to the professional programmer technologist. The ultimate concern is with an alteration of programming practices. Many changes will occur at the individual level where the single person is the adapter or rejector of an innovation. Change also occurs at the organization level where it may be called development, specialization, integration, or adaptation. Whatever it is called and on whichever level it occurs, change must ultimately be understood by a focus on the communication process and its participants.<sup>6</sup>

### PROGRAMMER CHARACTERISTICS

The programmer is a consumer of massive quantities of information. This individual must understand the problem environment by acquiring information through both formal and informal channels. The programmer must gather this information from internal and external sources as well as from a memory in order to develop a range of solutions to a problem.<sup>7</sup>

The programmer's purpose is to produce an acceptable piece of software. This goal is in sharp contrast to that of the computer scientist who is seeking a fundamental understanding of a concept and a resulting publication. The implications for the person supplying information to support both the programmer's function and the computer scientist's function are enormous. The scientist requires a systematic collection and organization of documents as well as a mechanism for making these documents available. Thus, the end product of one scientist (a publication) may serve as immediate input to a second scientist (Figure 1).

The programmer's output is a software product. In order for this to serve as input to a second programmer's task, the

program must be decoded and analyzed. The programmer clearly has access to the report of the scientist. However, the programming problem confronting this person is specific, immediate, and probably important to his professional and economic future (Figure 2). The scientist's report is abstract and as such will require interpretation; it is general and will therefore require translation to this specific problem; and it is unfamiliar to the programmer and thus the programmer using the results of this paper is likely to make a few mistakes at first.

It is much more efficient for the programmer to have another program and its programmer explain the software product. These products are used either as a basis for new programs or as prototypes of new programs. Documentation alone does not suffice. It often requires an explicit knowledge of the software product. Typically this requires human intervention or supplementation.

The programmer is faced with the dual problem of (1) following at least the trends in the ever increasing volume of information from scientists on software engineering, and (2) using mechanisms and/or people to aid in the decoding and analysis of already existing software products.

Keeping up with the available literature is an acknowledged problem among scientists. The growth rate of potentially relevant literature is increasing with no indication of achieving or approaching a limit.<sup>8</sup> This, of course, increases the difficulty of locating useful items among the useless items. Mooers<sup>9</sup> developed a "law" which states that information will not be used when it is more trouble to acquire the information than it is to be without the information. As a result, "information scientists" have developed information systems to help scientists meet their information needs. These attempts are documented and analyzed in the fourteen volumes of the *Annual Review of Information Science and Technology*. Unfortunately these systems are not adequate to meet the needs of the high level technologist such as the programmer.<sup>7</sup>

It was pointed out that programmers rely heavily on associations with other programmers. This does not occur without restrictions. Organizational constraints are placed on a programmer's ability to interact with those from other organizations. The programmer is expected to work with the employer to achieve this goal and is expected to refrain from preliminary disclosures. Thus, proprietary information inhibits the free flow of information among programmers.

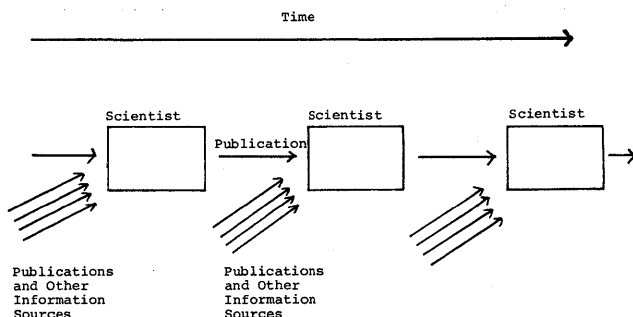


Figure 1—Scientists' development and use of information.

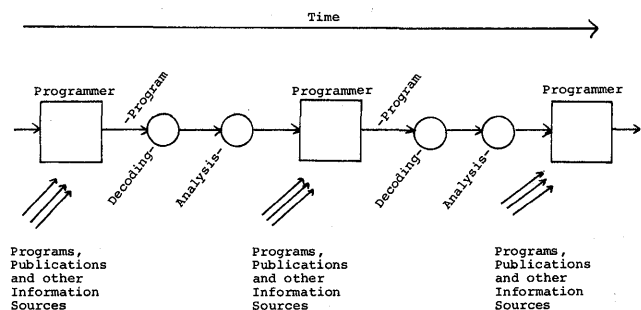


Figure 2—Programmers' use of information sources.

In spite of barriers, some techniques do seem to diffuse quickly among organizations. Allen<sup>7</sup> suggests that since humans are the best carriers of information, employee turnover may be a cause of the spread of technology. As an example, Roberts and Wainer<sup>10</sup> showed that applications for space technology occurred as individuals left the academic and space research organizations to start their own businesses or work for others.

For the truly effective transfer of technical information, one must make use of the human ability to recode and restructure information so that it fits into new contents and situations; each job change brings a record of experiences and a vast amount of "proprietary" information; a continual flow of job changes insures that no single firm is far behind.<sup>7</sup>

## INTEGRATION OF SCIENCE AND TECHNOLOGY

A common assumption is that science provides the basis for technology. However, Price<sup>11</sup> investigated patterns of citations among technologists and scientists over a 20-year time period to determine this dependence. He concluded that science and technology progress quite independently of one another. Gibbons and Johnston<sup>12</sup> tried to refute this by examining small technological advances. They did indeed trace five technological advances to a scientific source. The average time between the publication of the scientific finding and the technological advance was 12.2 years. There are examples of faster adaptations. If a need is developed by the technologist, then science will often attempt to fill the need and if successful a rapid use will be made of the scientific finding.

So there is some reason to believe that a connection exists between science and technology. In most cases, this is a slow process with occasional exceptions.

## IMPLICATIONS FOR SOFTWARE ENGINEERING

The realities of software engineering technology are (1) that the computer scientist is frustrated by the lack of adaptation of what are considered to be valuable tools and techniques, and (2) that the professional programmer is faced with the task of identifying information immediately relevant to a specific programming task; scientific papers are often viewed as abstract academic exercises with little practical application.

It is not surprising that one finds a variety of reasons why innovations have not occurred. For example, at a recent panel consideration on the lack of formal specification of programs, the following reasons were presented:

- . . . We lack adequate tools and support. We need better data base management facilities, better tools for viewing a system at different levels of abstraction, and better proof tools.
- . . . Progress has been slow because systems are hard to specify.
- . . . software development occurs over such a long time span (7-10 years) that both the software requirements and the state of the art in software specifications are likely to change.

- . . . Completely formal specifications are hard to write and people do not need them.
- . . . People fail in writing specifications because they approach it from only one viewpoint.
- . . . Progress is slow partly because the people working on specification techniques are not working on real systems. . . .<sup>13</sup>

A commonly heard prescription is that the programmer must be educated in the modern tools and techniques of software engineering! In fact, the proliferation of organizations, seminars and "experts" willing to reeducate professional programmers does provide evidence that many individuals and organizations recognize this as a need.

On the other hand, suggestions have been made to make the published papers available to the professional programmer more relevant to their immediate needs. For example Gerhart<sup>14</sup> suggests that the repeated publication of software engineering advances which use trivial or easily specified programs such as sorting or greatest common divisor is not a helpful practice for the professional programmer. It is a useful technique for the computer scientist in that it assists in the comparison of methods. However, the programmer ends up with a lack of either depth or breadth of experience with the programming tools. Gerhart suggests a publication outlet for free standing proofs of a variety of programs.

One might also suggest that information professionals have not done their jobs. There is little known about the information needs of the professional programmer. On the contrary one is more likely to read statements about what the professional programmer should be:

The world today has about a million "average programmers," and it is frightening to be forced to conclude that most of them are victims of an earlier underestimation of the intrinsic difficulty of the programmers task and now find themselves lured into a profession beyond their intellectual capabilities. . . . The conclusion that competent programming required a fair amount of mathematical skills has been drawn on purely technical grounds and, as far as I know, has never been refuted.<sup>15</sup>

But software engineering technologies will be accepted and used by even the most mediocre programmer if there is adequate motivation provided. Rogers and Shoemaker<sup>16</sup> note that:

The innovation process begins with an individual, or set of individuals, recognizing that their organization is facing a "performance gap" between their expectations and reality. This problem recognition sets off a search for alternatives, one of which may be innovation. The new idea usually comes from outside the organization, and must be matched with qualities of the organization's problem. Usually the innovation must be modified somewhat as it is implemented to fit the organization's conditions. So the innovation process consists of problem recognition, searching for alternative solutions, matching the innovation with the organization's problem, and implementation of the innovation, leading eventually to its institutionalization when it is no longer recognized as a separate element in the organization.

The information professionals must recognize the needs

of the professional programmer; the programming managers must recognize a need to create programs more effectively; professional programmers must be given the opportunity to communicate in the manner most effective for them to acquire the available tools and techniques; and adequate reward structures must be present to encourage change rather than inhibit it. There are no individuals at fault, and no simple prescription for change. Rather, there needs to be a recognition that the goals of the programmer are different from the goals of the computer scientist. The information useful to one is not useful to the other and the needs of the scientist are not the needs of the programmer.

## REFERENCES

1. Wasserman, Anthony I., "Problems of the '80's," Summary of Panel at International Conference on Software Engineering, May 1978, *ACM SIGSOFT Software Engineering Notes*, Volume 3, No. 3, July 1978, p. 29.
2. Gries, David, "Current Ideas in Programming Methodology," in G. Goos and J. Hartmans (ed.) *Program Construction: Lecture Notes in Computer Science*, Berlin, Germany, Springer-Verlag, 1979.
3. Yourdan, Edward and Constantine, Larry L., *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, New York, Yourdan Press, 1978.
4. Gries, David, "Current Ideas in Programming Methodology," in Peter Wagner (ed.) *Research Directions in Software Design Technology*, Cambridge, Mass., The MIT Press, 1979.
5. Hoare, C. A. R., "A Axiomatic Approach to Computer Programming," *Communications of the ACM*, Volume 12, October 1969, pp. 576-580, 583.
6. Rogers, Everett M., and Adhikarya, Renny, "Diffusion of Innovations: An Up To Date Review and Commentary," in D. Nimmo (ed.) *Communication Yearbook 3*, New Brunswick, New Jersey, Transaction Books, 1979.
7. Allen, Thomas J., *Managing the Flow of Technology: Technology Transfer and the Dissemination of Technology Information within the R & D Organization*, Cambridge, Mass., The MIT Press, 1977.
8. Brown, Harrison, "UNISIST: Growing Interest in a Worldwide Science Information System," *Journal of the American Society for Information Science*, Volume 22, No. 4, 1971, pp. 288-289.
9. Mooers, Calvin N., "Mooer's Law or Why Some Retrieval Systems are Used and Others Are Not," *Zator Technical Bulletin 136*, Cambridge, Mass., Zator Co., 1959.
10. Roberts, E. B. and Wainer, H. A., "Some Characteristics of Technical Entrepreneurs," *IEEE Transaction on Engineering Management*, EM-18, Volume 3, 1971.
11. Price, D. J. DeSolla, "Is Technology Independent of Science?" *Technology and Culture*, Volume 6, 1965, pp. 553-568.
12. Gibbons, M. and Johnson, R. D., "The Roles of Science in Technological Innovation," *Research Policy*, Volume 13, 1974, pp. 220-242.
13. Heninger, Kathryn L., "Limits to Specifications: Why Not More Progress?" Summary of Panel at IEEE Conference of Specifications of Reliable Software, April 1979, *ACM SIGSOFT Software Engineering Notes*, Volume 4, No. 3, July 1979, pp. 15-16.
14. Gerhart, Susan, "A Proposal for Publication and Exchange of Program Proofs," *ACM SIGSOFT Software Engineering Notes*, Volume 3, No. 1, January 1978, pp. 7-17.
15. Dijkstra, E. W., "On the Interplay Between Mathematics and Programming," in G. Goos and J. Hartmans (eds.) *Program Construction: Lecture Notes in Computer Science*, Berlin, Germany, Springer-Verlag, 1979.
16. Rogers, Everett and Shoemaker, F. Floyd, *Communication of Innovations: A Cross Cultural Approach*, second edition, New York, The Free Press, 1971.

# A technique for comparative assessment of software development management policies

by BRENDAN D. L. MULHALL and STEVEN M. JACOBS

*Jet Propulsion Laboratory  
California Institute of Technology  
Pasadena, California*

## INTRODUCTION

This paper describes a technique designed for organizing and structuring the comparison of software development management and software design practices. It is intended to provide a general method for assessing proposed software practices, especially of bidders on software contracts, and also to provide a visual aid in explaining to management how these proposed software practices comply with specifications, exceed specifications or are lacking.

An evaluator confronted with a request for proposal (RFP) or the proposal response to an RFP is often confronted with an enormous amount of documents that contain software policies, standards, and guidelines. Frequently, the software specification in the RFP or proposal response is not well organized and difficult to assess. To aid the evaluator in organizing his or her thinking and assure the completeness of the review, it is valuable to have a structured, disciplined approach to accomplish this evaluation.

This paper outlines a method of "getting started" with the evaluation process. It describes a technique for structuring the evaluation and illustrating completeness of software specification in the RFP proposal. This technique also aids reviewers of the final evaluation. Both the organization being evaluated and the evaluator's critiques and policy standards are checked.

The primary intention of this paper is not a summary of any of the referenced documents. Rather, this technique highlights those practices and policies which are considered valuable because they increase computer programmer productivity, reduce software life cycle costs (i.e., development and maintenance costs), and increase project management visibility. This effort was performed in support of the U.S. Army Remotely Piloted Vehicle (RPV) Project for the Aviation Research and Development Command (AVRADCOM), St. Louis, Mo, through the Defense Technology Office within the Technology Development Program at the Jet Propulsion Laboratory (JPL), California Institute of Technology. Consequently, U.S. Army and JPL software policies are used as an example, even though the assessment technique is completely general.

## ASSESSMENT METHODOLOGY

The technique is represented in the form of a matrix labeled Table I. The major row identifiers are nine key areas of computer software development methodology and management procedures. They are: Standards Required, Roles and Assignments, Documents, Planning, Testing, Reviews, Change Control, and Deliverables. Classical components of the software project life-cycle as defined by DeMarco [1] such as requirements analysis, design, coding, etc. are included within these rows.

There are three columns in Table I. A leading or identifying column is titled "Generic Names." This column categorizes the rows within each major category of the table. The generic names provide a consistent breakdown of the details of each major area of the software practice so that a complete evaluation is accomplished. Two blank columns follow which will be filled in as part of the policy assessment procedure.

The assessment of any software management policy is a straightforward procedure. First, a review of the software development policy under scrutiny is performed. Secondly, the policy to be used as a standard or baseline is broken down by generic name items and entered in its column. A column representing the bidder's policy is constructed and entries are made for each row (see Table I). Each item in the two columns is a specific detail of the two policies. The two policies are then compared, row by row. Concurrence with accepted software management policies as specified in a Request for Proposal, for example, can be determined by inspection, by noting where there are differences between the two columns. The degree of compliance is easily illustrated to any technical audience by use of the table. Shortcomings can also be highlighted. This degree of compliance can evolve into a more formal statement of adequacy of the given bidder's policy, to be used as a final decision point for award or modification of a software contract.

In evaluating proposals, each bidder's procedure would be compared row by row. Where there was no discussion, the bidder would be downgraded for lack of understanding. Where there is over specification, this would be noted as a

TABLE I.—Software development management assessment

Brendan D.L. Mulhall

	Generic Names	Accepted Standards	Bidders		Generic Names	Accepted Standards	Bidders		Generic Names	Accepted Standards	Bidders
STANDARDS* REQUIRED	Top-down Design, Structured Programming, Modularity, Documentation and Reviews, Document During Design, Test While Coding, Language, Firmware vs. Software			SOFTWARE DOCUMENTS*	Requirements, High-level Design, Detailed Design, Maintenance, Acceptance Test Procedures, Operations Manual			REVIEWS*	Requirements, Project Plan, High-level Design, Detailed Design, Acceptance Test Procedures, Operations Manual		
ROLES AND* ASSIGNMENTS	Source of Requirements, Task/Project Manager, Designer, Coder, Code Checker, Secretariat/Librarian, Sustainer, Acceptor, Tester, User/Operator, Consultant, Configuration Management			PLANNING	Responsibility, Document, Estimating, Scheduling, Budgeting, Resource Allocation, Computer Resources, Project Control			CHANGE* CONTROL	During Development, After Transfer to Operations		
				TESTING	Top-Down Test, Design of Test, Test Data, Test Tools, Test Results, Error Handling, Verification and Validation, Acceptance, Integration, In-Plant and Field Test			DELIVERABLES*	Documents, Code, Test Results, Test Data		

\*Software Development Management areas that require particular attention

possible cost reduction during the implementation or as a result of contract negotiation. Thus, Table I is not merely a comparison of management policies of two selected software development policies. Rather, Table I can be expanded and used as a tool for analyzing any number of software development tasks, comparing given management practices with a variety of accepted standards and the software engineering literature.

APPLICATION OF THE TECHNIQUE

An example of the application of the technique is shown by the matrix contained in Table II. The first two columns of the matrix are two U.S. Army procedures for software design practices and management policy. The first is a proposed procedure [2] (Army regulation) and the second, the procedure specified in the RPV Request for Proposal, [3,4].

The next two columns are software procedures which have been used at one time or another at JPL. These are the procedures for the National Aeronautics and Space Administration (NASA) Deep Space Network (DSN) [5-11], and the

Department of Energy sponsored Vehicle Economy, Emissions, and Performance (VEEP) computer simulation project [12]. Finally, the last two columns are titled "The Literature" and show four reference texts which represent a sample of the academic viewpoint written by Tausworthe [13,14] and Yourdon [15,16].

In some instances, the generic names simply do not apply across the board and, consequently, a bracket is used to communicate the collapse of these identifiers. In this case, a short description is used instead which is amplified in the text.

The nine key areas of software development and their concomitant generic names are described below as are the entries for the four policies and four texts used for the example.

Standards required

In this major area, those standards which are fundamental to modern software engineering according to Tausworthe<sup>13,14</sup> and Jensen and Tonies<sup>17</sup> were listed. These are top-down design, structured programming, modularity, documenting

TABLE II. Example of software development management assessment

Brendan D. L. Mulhall

	Generic Names	U. S. ARMY		J P L		THE LITERATURE	
		Proposed	Specified in RPV RFP	Deep Space Network	VEEP	Tausworthe	Yourdon
STANDARDS* REQUIRED	Top-down Design,	Required	Required	Required	Followed Yourdon's Standards	Top-down design, Structured programming, Modularity, Documentation, Concurrent documentation, Requirement and definition, Design and specification, Coding, Testing, Quality Assurance	Top-down design and implement, Structured design, Structured programming, Structure charts, Chief programmer teams, Program librarians, Structured walkthroughs, Pilot projects
	Structured Programming,	Required	Required	Required			
	Modularity,	Required	Required	Required			
	Documentation and Reviews,	Partially specified	Required	Specified in detail			
	Document During Design,	Omitted	Omitted	Strongly encouraged			
	Test While Coding,	Omitted	Omitted	Strongly encouraged			
	Language,	Omitted	High level	High level			
	Firmware vs. Software	Omitted	Same methodology for both	Same methodology for both			
ROLES AND* ASSIGNMENTS	Source of Requirements, Task/Project Manager, Designer, Coder, Code Checker, Secretariat/ Librarian, Sustainer, Acceptor, Testor, User/Operator, Consultant, Configuration Management	Needs to be defined	To be defined in (contractor provided) Computer Program Development Plan (CPDP)  Assigns code check to programmer	Subsystem Requirements Engineer	Cognizant Engineer Chief Programmer Entire team Support Programmer Librarian Librarian Sponsor Programmers Cognizant Engineer Cognizant Engineer Cognizant Engineer Librarian	Project Manager, Chief Program Designer, Lead Programmer, Test Engineer, Interface Control Engineer, User Representative	Analyst, Designer, Chief Programmer, Copilot, Administrator, Editor, Secretary, Librarian, Toolsmith, Language Lawyer, Programmer, Tester
	Identified						
	Cognizant Development Engineer						
	Identified						
	Quality Assurance						
	Identified						
	Cognizant Sustaining Engineer						
	Cognizant Operations Engineer						
	Cognizant Development Engineer						
	Cognizant Operations Engineer						
	Not specified						
	Secretariat						

with design, testing during coding, the preferred use of high-level languages, and standards for firmware. The applicability of each of these is indicated across the row which is intended to show that it is a recognized attribute of the particular standard practice.

Roles and assignments

In this area, the concepts and semantics become something of a problem because every organization has its own names. For example, the DSN lists Cognizant Design En-

TABLE II. Example of software development management assessment (continued)

	Generic Names	U. S. ARMY		J P L		THE LITERATURE	
		Proposed	Specified in RPV RFP	Deep Space Network	VEEP	Tausworthe	Yourdon
SOFTWARE DOCUMENTS*	Requirements,	System specification	To be defined by contractor in CPDP	Software Requirements Document	In Research Objectives Plan	Software Requirements Document	Required
	High-level Design,	Omitted		Software Design Document	Design document	Software Design Document	Structured design
	Detailed Design	Development specification		Software Specification Document	Design document	Software Specification Document	Procedural design
	Maintenance,	Product specification		Software Specification Document	Documentation	Maintenance manual	Development Support Library
	Acceptance Test Procedures, Operations Manual	Coordinated test program Operators manuals		Software Test and Transfer Doc	Technical paper	Software Test Report	
				Software Operating Manual	User guide	User manual	
PLANNING	Responsibility Document, Estimating, Scheduling, Budgeting, Resource Allocation, Computer Resources, Project Control	Computer Resource Management Plan (CRMP)	To be defined by contractor in CPDP  Speed, memory, & CPU reserve in CPDP	Cognizant Development Engineer	All planning performed by Task Manager	Schedule established with phased concurrency of development activities	Complete management procedure for each version or major modification
	Specified in DSN Software Management and Implementation Plan						
TESTING	Top-Down Test, Design of Test,	Omitted	Omitted	Required	Required	Omitted	Required
	Test Data,	Required	Required	Required	Required	Required	Omitted
	Test Tools	Omitted	Interpret inputs	Required	Required	Required	Omitted
	Test Results,	Required	Test Materials	Required	Required	Required	Required
	Error Handling,	Monitoring	Omitted	Required	Required	Required	Omitted
	Verification & Validation	Required	Required	Required	Required	Required	Required
	Acceptance,	Required	Required	Required	Required	Required	Omitted
	Integration,	Required	Omitted	Required	Required	Required	Required
	In-Plant and Field Test	Omitted	Required	Omitted	Omitted	Omitted	Omitted

TABLE II. Example of software development management assessment (continued)

Generic Names	U. S. ARMY		J P L		THE LITERATURE			
	Proposed	Specified in RPV RFP	Deep Space Network	VEEP	Tausworthe	Yourdon		
REVIEWS*	Requirements,	System Specification,	CPDP	Once	In Research and Technology Objectives and Plans (RTOP)	Same as DSN with critical software acceptance	During structured walkthroughs	
	Project Plan,	CRMP	CPDP	Monthly work breakdown structure				
	High-level Design,	Development specification	Omitted	Module by module				Module by module,
	Detailed Design,	Development specification	Module by module	Module by module				When requested,
	Acceptance Test Procedures,	System acquisition	CPDP	Once				At sponsor delivery,
Operations Manual	CRMP	CPDP	Once	Via user guide				
CHANGE* CONTROL	During development,	None	To be defined by contractor in CPDP	Controlled by software implementation team and maintained by secretariat	Team agreement on a module basis	Change control cycle established when problem occurs affecting software development library elements under configuration control	During structured walkthroughs	
	After transfer to operations	Software Configuration Control Board	In CPDP	By formal change control	Sponsor's responsibility with support		Omitted	
DELIVERABLES*	Documents,	All software documents	All software documents	All documents except SRD and SDD	All documents	Recommends preparing a checklist of deliverable software items	Omitted	
	Code,	Software and operating system	High level language and machine code	High level language and machine code	High level language and machine code			
	Test Results, Test Data	According to approved test plan	Required Required	Required Required	In technical paper In technical paper			
SOFTWARE DEVELOPMENT COSTS	None	N/A	TBD	\$8 million (Mark III Implementation)	\$250,000	N/A	N/A	

\* Software Development Management areas that require particular attention

gineer (CDE), Cognizant Sustaining Engineer (CSE), and Cognizant Operations Engineer (COE) for the key software tasks of design, maintenance, and operation, respectively. Under "The Literature," Yourdon has several interesting and even amusing names for roles which are identified in other systems. This text identifies roles which exist in every computer facility though perhaps not explicitly recognized, such as "Tool Smith" and "Language Lawyer."

Discussion of common practice with DSN personnel was required to determine the equivalents to the "Tool Smith," in their policy of requiring a single development of utility software (such as particular I/O handlers or character-code conversions) that will meet identical requirements in a number of programs. On the other hand, the function performed by Yourdon's "Language Lawyer" is often not centralized, but is duplicated many times over by programmers who individually maintain sizable files of memos and other correspondence that tell how the operating systems and compilers really work.

The effects of these differences in terminology cause significant expenditure of effort to determine the real correspondence between role statements, efforts that are not permitted in the evaluation of competitive bids. This is a

problem area in the use of the technique described here and requires considerable judgment by the users of the technique.

#### Software documents

Adequate documentation of software is imperative. The Documents row outlines the names of required documents that accompany software development. For example, the DSN requires the Software Requirements Document (SRD) which contains a statement of those system functional requirements which are to be implemented in software, Software Design Document (SDD) the top-level architectural design, Software Specification Document (SSD) which contains the detailed design and which becomes a maintenance document, Software Test and Transfer Document (STT) for describing test and transfer to operations procedures, and Software Operations Manual (SOM) the user guide. Other projects require Software Functional Description (SFD), which includes functional requirements, and Software Test Report (STR), which document test results as well.

There is a conscious effort in the selection of generic document names in the table to make them relate in a nearly

one-to-one way with the Roles and Assignments. This is to indicate the author or responsible person for each document. This correspondence is further carried on in the Reviews section and could become a third dimension in the table.

### *Planning*

Planning is, of course, a key step in any project implementation, and imperative in software development. The approval of a well-defined software management and implementation plan is a common practice at JPL and essential for software development management. Even small software tasks require planning in order to meet schedule and functional requirements. Two items of planning, namely resource allocation (the employment of personnel and other resources that we measure in dollars such as computer time, graphic support, etc.) and computer resources, (the amount of core, CPU time, peripherals, etc. which any program shares with other programs while co-existing in the same software effort), may seem redundant but are not.

In the allocation of resources, the life cycle cost of the system is of primary importance. The life cycle costs consist of the design and implementation costs as well as the operating and maintenance costs through the useful life of the system, and, finally, archival storage costs after replacement. To forecast life cycle costs one must consider both the software and the hardware together as a complete operating system. For example, if a program must be squeezed down by partitioning into segments to fit into a limited amount of main memory, as opposed to designing with an adequate amount of memory to reasonably accommodate the program and also to allow for some growth, then both implementation and maintenance costs may be increased. Similar tradeoffs may exist such as I/O and the choice of peripherals.

### *Testing*

Ultimately, all useful computer programs must be shown to be both correct and valid. A "correct" program has internal consistency within its architecture, logic, syntax, and nomenclature. A valid program satisfies the functional requirements of the user.

Like top-down design, top-down testing is often, but not always, according to Yourdon and Constantine<sup>18</sup>, employed to gain the full benefit of the top-down approach. Procedures for testing software must be designed in advance. Definitions of test data, test tools, and the use of stubs as in Yourdon<sup>15,16</sup> are also required. Procedures to be followed based upon the test results must also be defined. The terms verification and validation must clearly be understood by both the software bidder and customer.

Structured walk-throughs, as described by Yourdon [15,16], are a key means of ensuring software consistency and correctness. The criteria for software acceptance must also be established. The integration of a software system requires additional test procedures, occasionally examining software

in an in-plant versus field use test, or over the entire life cycle of the system.

### *Reviews*

This section uses the same generics as the documents and the roles and assignments sections to show this relationship.

### *Change control*

Any change to computer software can have drastic effects upon delivery schedule and computer program validity. Changes that occur during or after the software design and implementation require change control procedures. Therefore, the comparisons of change control procedures in Tables I and II are divided into two areas, the first being the change control during the development period and secondly, after initial acceptance of the software by the user. These are separate and very distinct areas requiring different procedures. Changes that affect interfaces being worked on by different groups require careful control at all times after the interface definition has been established, even during implementation. Changes that are found necessary after design review of architectural or high-level designs need not be as formally controlled since the software has not been accepted by the user and disseminated through the user organization. Authorization of the change at this stage still requires the same approval cycle as the original design.

When properly applied, top-down design, modularity, and structured programming techniques all help to minimize or prevent the change from rippling through the system design by isolating each function in its own module and by eliminating sneak paths which allow changes to cause unforeseen effects.

An automated software development tool such as SDDL (Software Design and Documentation Language), developed at JPL, can aid in tracking the changes that occur during the design process itself by making it easy 1) to record such changes and 2) to see where changes have occurred. SDDL also lends itself to software implementation using structured programming techniques, minimizing the effects of changes when structured, modular computer programs are constructed [19].

Changes that occur to software after delivery require formal change control procedures that not only update the software, but also inform others by issuing a notification of change and change pages to holders of the software documentation and, update operators' manuals.

### *Deliverables*

Not all projects call for the same items as deliverables. For example, the requirements and the high-level design documents are not part of the standard deliverables for the DSN because they were not deemed necessary to survive the implementation. Also, not every standard practice calls



for delivery of both machine code and high-level language though this is an essential item for maintenance purposes. Operating systems are often required as part of software delivery. The provision of operations and maintenance manuals also are handled differently by different projects.

In the item with the generic name Code, there are two levels specified. High-level language or source code can be a program design language or a compiler language such as FORTRAN or COBOL. Even assembly language, if that was the highest level language used in the development, can be the source code. The most human-oriented language employed is the source code and should be included as part of the product delivered to the customer, since it is needed to maintain and revise the program. The second level is machine code representing the entire working program.

#### *Software development costs*

This last row in Table II indicates the relative magnitude of the software implementation projects. It is recognized that complete, rigorous software practices are not always appropriate in small projects. The most efficient approach is to use those aspects of software methodology which are valuable for control, design, maintenance, and survivability of the software system while not requiring all of the formal reviews, documentation standards, etc. which might be required to manage a larger project. Therefore, keeping in mind the size and scope of the programming task while utilizing the assessment technique is important.

If more than two procedures are being compared, relative development costs become an interesting item and, hence, are included in Table II but not Table I.

#### COMPARISON OF POLICIES

An inspection of Table II indicates the comparability of the six approaches to software management and design. The most similar are the Deep Space Network Policy and the texts by Tausworthe which is due to the contribution that Tausworthe made in developing DSN standard practices. The Vehicle, Economy, Emission, Performance program was based heavily on the texts by Yourdon and, consequently, is a specific application of these texts.

The two Army procedures do differ somewhat. The specified procedures in the RPV RFP are based on two references, one (the RFP) is a collection of statements and various Army documents which formed the specification. The proposed Army practice [2], is more of a computer hardware resource management procedure, although it has a number of requirements on software procedures. Consequently, these two columns are not entirely comparable, but were included to show both the present Army philosophy and one possible future approach.

One area of comparison between proposals and standard practices which is not delineated in the table, but is to the advantage of the software bidder is the clarity, conciseness,

and discernibility of statements of the required procedures and practices.

#### A FINAL NOTE

Due to the differences in software policies and guidelines combined with the lack of maturity of the field of software engineering, a fair and just comparison of a number of software proposals or policies can be difficult. One must gain a substantial familiarity with the proposals or policies to be evaluated to secure valid comparison data for this technique. Special treatment of information in regards to particular proposals or policies is often necessary and slight modifications of the standards displayed by this technique may be in order. A weighting factor may also be incorporated into the table by adding another column to Table I for a multiplier or weighting-type function, or simply being more critical of certain areas and not in others, in a more subjective approach.

When there is adequate interaction between the software practitioners and a group of knowledgeable evaluators, the assessment technique can be a useful tool for highlighting the differences between approaches and for providing a basis for determining how approaches could be modified or improved.

#### CONCLUSION

Any bidder's proposal can be evaluated by constructing a new column and filling in each area the names or the existence or the absence of certain documents, reviews, assignments of personnel, etc. By this means the conformity of the proposed standard practice to existing conventional wisdom can be related. Shortcomings and better procedures in certain areas can also be readily identified and communicated.

The assessment technique can assist in giving structure to the process of comparing bidder's proposals, management policies of different projects or organizations, or to evaluate proposals for software management approaches.

#### ACKNOWLEDGMENT

This paper presents the results of one phase of research conducted at the Jet Propulsion Laboratory, California Institute of Technology for the United States Army Aviation Research and Development Command, by agreement with the National Aeronautics and Space Administration.

#### BIBLIOGRAPHY

1. DeMarco, Tom, *Structural Analysis and System Specification*, New York: Yourdon, Inc., 1979.
2. *Management of Computer Resources in Army Defense Systems*, U. S. Army Publication #AR 70-XX, March 1978.
3. *Research and Development Software Acquisition, A Guide for the Material Developer*, U.S. Army Publication #AMCP 70-4, September 1974.

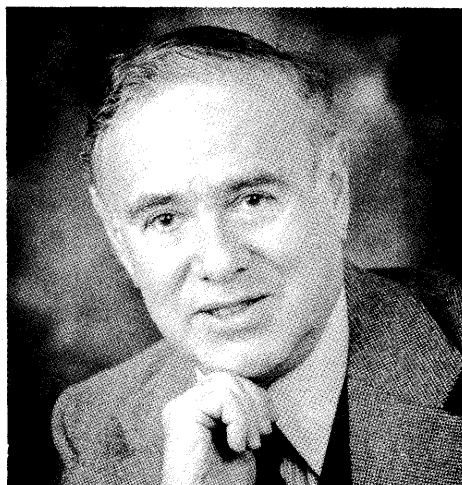
4. *Request for Proposal for U.S. Army Remotely Piloted Vehicle Target Acquisition/Designation Aerial Reconnaissance System*, U.S. Army RFP #DAAK50-78-R-0008, May 1978.
5. Irvine, A. P., Editor, *Standard Practices for the Implementation of Computer Software*, JPL Publication #78-53, September 1978.
6. *Software Implementation Guidelines and Practices*, JPL DSN Standard Practice Publication #810-13, August 1975.
7. *Preparation of Software Requirements Documents*, JPL DSN Standard Practice Publication #810-16, December 1975.
8. *Preparation of Software Definition Documents*, JPL DSN Standard Practice Publication #810-17, July 1976.
9. *Preparation of Software Specification Documents*, JPL DSN Standard Practice Publication #810-19, March 1977.
10. *Preparation of Software Operator's Manuals*, JPL DSN Standard Practice Publication #810-20, February 1977.
11. *Preparation of Software Test and Transfer Documents*, JPL DSN Standard Practice Publication #810-21, November 1976.
12. Vehicle Economy, Emissions, and Performance Program Development for the *JPL Automotive Technology Status and Projections Project Final Report*, JPL Publication #78-71, June 1978.
13. Tausworthe, Robert C., *Standardized Development of Computer Software, Part I, Methods*, JPL Publication #SP 43-29, July 1976.
14. Tausworthe, Robert C., *Standardized Development of Computer Software, Part II, Standards*, JPL Publication #SP 43-29, Part II, August 1978.
15. Yourdon, Ed, *How to Manage Structured Programming*, New York City: Yourdon, Inc., 1976.
16. Yourdon, Edward, *Techniques of Program Structure and Design*, Englewood Cliffs, N.J.: Prentice-Hall, 1975.
17. Jensen, Randall W. and Tonies, Charles C., *Software Engineering*, Englewood Cliffs, N.J.: Prentice-Hall, 1979.
18. Yourdon, Edward, and Constantine, Larry L., *Structured Design: Fundamentals of a Discipline of Computer Program and System Design*, Englewood Cliffs, N.J.: Prentice-Hall, 1979.
19. Kleine, H., *Software Design and Documentation Language*, JPL Publication #77-24, July 1977.



## Software Reliability

The first session of the software reliability area will address Software Reliability Needs. It includes three invited papers that deal, respectively, with the origination of reliability requirements, with issues of reliability measurements, and with reliability modeling and prediction. All of them represent the cutting edge of the current technology and treat the subject in a broad manner that may be of interest also to the non-specialist; e.g., the discussion of reliability requirements is based on software used for a fly-by-wire system for transport aircraft. In this context, the overall aircraft safety regulations are examined, and the derivation of computer and software reliability requirements from these is outlined. All papers recognize that software reliability is an evolving discipline, one in which we are all students, and a field of tremendous importance for the future of computers in our society.

The second session addresses Current Trends in Software Reliability, and it is structured as a panel discussion. The panelists work in fields that make utmost demands on the reliability of computing systems, and they will relate the current practices that are being used to meet these requirements. Management techniques, personnel selection and training, and technical methods in requirements analysis, design practices, structured programming and innovative test strategies will be discussed.



Herbert Hecht  
*Area Director*



# Standard error classification to support software reliability assessment

by JOHN B. BOWEN

*Hughes-Fullerton*  
Fullerton, California

## SUMMARY

A standard software error classification is viable based on experimental use of different schemes on Hughes-Fullerton projects. Error classification schemes have proliferated independently due to varied emphasis on depth of casual traceability and when error data was collected. A standard classification is proposed that can be applied to all phases of software development. It includes a major casual category for design errors. Software error classification is a prerequisite both for feedback for error prevention and detection, and for prediction of residual errors in operational software.

## INTRODUCTION

The ability of managers and technical developers to influence the reliability of software is very high at the outset of a project but declines rapidly as commitments are made, schedule time and budgets are used, and code and documents are produced. The acceptance test phase is the very time when little chance remains to influence the reliability of the system except by rebuilding the deficient parts. A significant goal is to alert management as early as possible in the development phases of critical problems and adverse trends that could degrade software reliability. Since up to 60 percent of the errors detected in the life cycle of software have been committed during the design phase,<sup>1</sup> a major challenge is to devise error categories that are sensitive to that phase, and thereby provide feedback. Management feedback has been difficult to obtain, because programmers have traditionally enjoyed a pride of codemanship that rarely admits to the existence of errors. However, with the advent of Modern Programming Practices (MPPs), such as code reviews, software errors are available for analysis and feedback—even before a program module is executed.

A special conference on the problems of data collection<sup>2</sup> concluded that "The most success in data collection has been realized in those places where there has been feedback." Over three years ago Marcia Finfer<sup>3</sup> noted that "Many papers addressing the problem of error collection and quantization state that greater understanding of software errors will lead to the improvement in the design and

application of software development tools and techniques, but the reality of the situation does not support this conclusion. Project managers who have the ability to both initiate error reporting procedures, and analyze the incoming data, do not consistently take action resulting from the analysis of the error reports." This observation is still true today.

Typically, the reason for not acting on the analysis of error trends is the overriding pressure of getting the immediate job done on schedule. Such a reason is understandable particularly during the latter stages of development. However, software management appears to be remiss in not applying the results of error analysis to subsequent projects.

In the case of predictive software reliability models, most program managers have doubts about their usefulness. Consequently some view error data collection as a nonproductive extra burden. This view is unfortunate, because only with the support of conscientious error data collection can proposed quantitative reliability models be validated.

The Rome Air Development Center (RADC) has sponsored numerous studies on software error collection and analysis, starting with a software reliability study by TRW<sup>1</sup> which included an error category scheme, which was generated as the raw error data was analyzed. This error scheme was used in later RADC studies,<sup>4,5</sup> but no approved standard error classification has emerged within the Air Force, to date. The Navy has included a software trouble classification in its recent MIL-STD-1679;<sup>6</sup> however the four categories do not have enough detail to assist in feeding back constructive information to management.

This paper proposes the standardization of a set of software error classifications that have casual, severity, and source phase properties. Such a set will assist the project manager in taking remedial action to improve reliability, support company and software community efforts in evaluating the impact of reliability-producing techniques, and aid in validating software reliability models and metrics. The term software reliability, therefore, as used in this paper represents both the assessment of the use of reliability-producing factors and the prediction of residual errors. Although reliability models are primarily used to predict residual errors existing after acceptance testing, they can also be applied to earlier development phases if primed with sufficient error data. Reliability prediction is not concerned with the casual

properties of errors, but should be concerned with severity and source phase properties.

### NEED FOR A STANDARD ERROR CLASSIFICATION

Like most human activities, the software engineering environment is a complex of a great variety of interrelated factors. Some researchers such as Willmorth, et al.<sup>7</sup> conclude that "No one set of data parameters collected for research purposes will significantly support a wide range of reliability analyses." Weiss<sup>8</sup> contends that error classifications need to be tailored for each study or application so that the questions of interest can be answered. I contend that there is a need for a standard scheme to classify error data which represents the basic characteristics of the software environment.

In fact, a number of organizations and agencies such as the Joint Logistic Commanders, U.S. Navy, IEEE Computer Society, and a number of industrial companies have developed, or are in the process of standardizing, software error classifications. Unfortunately, few of these schemes are compatible with each other. Only the severity classifications are similar, and even in this case the number of severity categories ranges from three to five. RADC has inaugurated a software data collection and analysis program<sup>9</sup> which has as one of its major objectives to "Promote standards of software data collection, and support the development and definition of common software data collection terminology."

The necessity of a standard error classification scheme becomes evident when the needs of a large project and research activities are examined. A few examples are: to provide feedback to develop software design standards; provide guidance to test engineers; evaluate modern programming practices; evaluate verification and validation tools; and validate and support quantitative reliability models. The minimal ingredients of such a scheme are listed in Table I.

Since some studies report that as much as 60 percent of all software errors originate in the design phase, it is important that error collection and classification be sensitive to the point in time in the life cycle of a program when the error occurs. Only then can improved software design standards be developed. In addition, the distribution of types of errors from related projects can assist test engineers and quality analysts in concentrating their activities. For instance, if one particular application is expected to have a preponderance of computational errors then the test planners would profit by applying dynamic tools, rather than static tools, to uncover such errors.<sup>10</sup> Thus, while it has been

established that the use of error classifications can aid in evaluating all phases of software development, the most rewarding efforts occur during the early phases, such as design. As suggested by Finfer,<sup>3</sup> error analysis can indicate the necessity to apply additional personnel to a particularly error-prone program or subsystem, and a cluster of errors in a related group of programs may indicate that particular software is poorly designed. In a study for NASA-Langley, Hecht<sup>11</sup> recommended that "Classification by cause of failure is desirable in order to organize remedial measures. This information is of value for the management of the immediate project on which it is obtained, for overall software management (e.g., in guiding the allocation of resources), and for the development of improved software engineering tools and procedures (language processors, test tools)."

Thus while these examples illustrate the underlying necessity for developing a standard software error classification scheme, the problem is not exactly new. A software data collection conference in 1975<sup>2</sup> concluded that: "Standardization of data items, collection procedures, and project characteristics is needed to provide comparability of measures in evaluating tools, techniques, and methods." This is still true today especially in the validation of predictive software reliability models and software reliability metrics, as well in the selection of the best V&V tools and techniques.

One of the major hurdles in comparability is the difficulty in controlling all of the factors that influence software development during an experiment that compares two software development activities using different modern programming practices. It is difficult to compare the programming activity of different projects using error analysis because of uncontrollable factors—such as programmer background, hardware and software environment, and applications. Error density is frequently used to evaluate MPPs. For example, IBM<sup>12</sup> compared two large projects: One project with top-down design, structured code, chief programmer teams, and a librarian, had an error rate of 1.0 per 100 lines of code. Another project, using conventional techniques, had twice that rate. This report is an example of the typical use of unqualified errors to evaluate the effectiveness of MPPs. In the final analysis, such a use can be misleading unless the researcher reveals when the errors were detected, and how severely these errors impact mission performance. Even if the errors are qualified there must be a common understanding of the classification scheme. Susan Gerhart<sup>13</sup> reports "The study of observed errors on the fallibility of modern programming methodologies suffered from an inconsistent error domain which caused several types of classification schemes to be difficult to construct and to interpret."

Castle, in a thesis on validation of software reliability math models,<sup>14</sup> states that if he had to make one recommendation, it would be the importance of continued software error collection. He pointed out, "A disease cannot be cured without knowledge of the cause. So is the case with unreliable software." In a list of 22 software error characteristics for collection, he includes the phase in which the error occurred, the criticality of the error, and the error categories (causal) with unambiguous definitions. As a result of a study of candidate software reliability models, Kruszewski<sup>15</sup> recom-

TABLE I.—Questions that can be answered by a feedback-oriented classification scheme

When	-	In what phase in the software development cycle did the error originate?
How	-	What did the designer/analyst/programmer do wrong?
What	-	What is the effect of exercising the resultant fault?

mended improved data collection with formal error reporting and using causal and severity categories. Schafer in a recent RADC study to validate candidate software reliability models<sup>16</sup> used 16 sets of project error data which represented a total of 31,181 errors. The results of the study indicated that in general the software models fit poorly due to vagaries of the data, rather than shortcomings of the models. The study report concluded that more work remains in the area of software error data collection. Echoing these findings, Sukert, at a recent conference,<sup>17</sup> recommended the development of software error data collection standards, and the study of software reliability predictions based on error criticality categories.

#### SURVEY OF CANDIDATE ERROR CLASSIFICATION SCHEMES

An excellent survey of the state-of-the-art in software error data collection and analysis was published by Robert Thibodeau.<sup>18</sup> His report describes recent efforts of government agencies, educational institutions, and private companies; and includes synopses of several studies on software error collection and analysis. On the topic of error classification he states:

"The study of software errors requires them to be separated according to their attributes. This is the first step in understanding what causes them and, subsequently, how they may be prevented. The need for a *practical* error classification is important and, since it applies to nearly all areas of software research, it deserves to be treated as a separate topic."

#### Mitre error classification study

In early 1973 MITRE Corporation, under contract to RADC, developed a general software error classification methodology.<sup>19</sup> The methodology was designed to serve as a guideline for experiment-specific application. The proposed classification scheme is hierarchical, and consists of five major categories:

1. Where did the error take place
2. What did the error look like
3. How was the error made
4. When did the error occur
5. Why did the error occur

The associated subcategories are not unique to the major categories and include attributes such as People, Hardware, Software, Mechanical, Intellectual, and Communicational. The scheme accounts for the fact that a single error can have a number of characteristics occurring simultaneously. The report addresses the problem of multiple classification of the same error, and suggests the use of the fuzzy set theory where multiple classifications are qualified by degree to fully describe a single software error.

#### TRW software reliability study

During a study for RADC,<sup>1</sup> TRW-Redondo Beach devised a software error classification scheme with twelve major causal categories. The study also developed a source phase classification. These classifications which were iteratively developed during a 2.5-year study are listed in Table II.

#### Study of errors found in validation

Raymond Rubey in a technical paper published in 1975<sup>20</sup> presents several error categories. He stated that, "The most basic data required about the errors found during validation are the frequency of occurrence of those errors in defined error categories and their relative effect or severity." Three of the proposed error classification schemes are included in Table III.

#### AN/SLQ-32(V) verification and validation

In May 1977 the Navy distributed a statement of work for V&V services<sup>21</sup> which characterized the software errors encountered during software development as follows:

Requirements  
 Processing Design  
 Data Base Design  
 Interface Design  
 Processing Construction  
 Data Base Construction  
 Interface Construction  
 Verification  
 Specification (all documentation)

TABLE II.—Software error classifications developed during TRW reliability study

<u>Causal</u>	
	COMPUTATIONAL
	LOGIC
	DATA INPUT
	DATA HANDLING
	DATA OUTPUT
	INTERFACE
	DATA DEFINITION
	DATA BASE
	OPERATION
	OTHER
	DOCUMENTATION
	PROBLEM REPORT REJECTION
<u>Source Phase</u>	
	REQUIREMENTS
	DESIGN
	CODING
	MAINTENANCE
	NOT KNOWN



TABLE III.—Error classifications proposed by Rubey study

<u>Causal</u>	
INCOMPLETE OR ERRONEOUS SPECIFICATION	
INTENTIONAL DEVIATION FROM SPECIFICATION	
VIOLATION OF PROGRAMMING STANDARDS	
ERRONEOUS DATA ACCESSING	
ERRONEOUS DECISION LOGIC OR SEQUENCING	
ERRONEOUS ARITHMETIC COMPUTATIONS	
INVALID TIMING	
IMPROPER HANDLING OF INTERRUPTS	
WRONG CONSTANTS AND DATA VALUES	
INACCURATE DOCUMENTATION	
<u>Severity</u>	
SERIOUS	
MODERATE	
MINOR	
<u>Source Phase</u>	
DEFINING THE PROGRAM SPECIFICATION	
DEFINING THE PROGRAM	
CODING	
PERFORMING MAINTENANCE FUNCTIONS	

*JLC preliminary error classification*

In April 1979 the Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resource Management held a software workshop<sup>22</sup> where preliminary general categories for classifying software errors were defined. As shown in Table IV three major casual categories and four severity categories were included.

TABLE IV.—Software error categories proposed by Joint Logistic Commanders

<u>Software Specifications</u>	
1. Unnecessary functions	
2. Incomplete requirements or design	
3. Inconsistent requirements or design	
4. Untestable requirements or design	
5. Requirements not traceable to higher specifications	
6. Incorrect algorithm	
7. Incomplete or inaccurate interface specifications	
<u>Code</u>	
1. Syntax errors	
2. Non-compliance with specification(s)	
3. Interface errors	
4. Exception handling errors	
5. Shared variable accessing error	
6. Software support environment errors	
7. Violation of programming standards	
8. Operational support environment errors	
<u>Data</u>	
1. Accuracy	
2. Precision	
3. Consistency	
<u>Severity</u>	
1. Prevents accomplishment of its primary function, jeopardizes safety, or inhibits maintainability of the software	
2. Degrades performance or maintainability, with no workaround	
3. Degrades performance or maintainability, but a workaround exists	
4. Doesn't adversely affect performance or maintainability (such as documentation, etc. errors transparent to users)	

*Discussion*

Most of the software error classification schemes surveyed have a separate classification for severity or impact on mission performance. However, there was no general agreement on using distinct classifications for cause and source phase. Some error causes are phase peculiar, therefore a combined single category would result in fewer subcategories than all possible combinations of source phase and casual subcategories. This advantage appears to be outweighed, however, by the ease in implementing automated statistical analysis of the phase and casual attributes when the categories are separated.

It should be noted that only the Navy AN/SLQ-32(v) casual classification scheme included unique categories for design errors. Rubey's classification contains only one special design category, intentional deviation from specification. (This category could be interpreted as representing either a design or coding activity.) The JLC classification has design categories; however they are combined with requirements (e.g., incomplete requirements or design).

## RESULTS OF USING EXPERIMENTAL CLASSIFICATIONS ON HUGHES-FULLERTON PROJECTS

For over two years Hughes experimented with a software error classification scheme on an Army project during the development phases. The classification scheme used on this project was based on the scheme proposed by Rubey.<sup>20</sup> Three classifications were used: Severity, Cause, and Miscellaneous as shown in Table V. The casual classification

TABLE V.—Hughes-Fullerton experimental error classification

<u>Severity</u>	
CR	System Crash or Serious Effect on Mission Performance
MA	Incorrect Values that Reduce Mission Performance
MI	Incorrect Values that have Tolerable Effect on Mission
<u>Cause</u>	
REQMT	Expanded, Reduced, or Erroneous Requirements
PROGM	Non Responsive Program Design
SPECS	Incomplete or Erroneous Program Design Specifications
LOGIC	Erroneous Decision Logic on Sequencing
IMPVE	Improved Program Storage or Response Time
INTRT	Improper Handling of Interrupts
LINKE	Incorrect Module or Routine Linkage
ARITH	Erroneous Arithmetic Computations
ALGOR	Insufficient Accuracy in Implementation of Algorithm
DOCUM	Inaccurate or Incomplete Comments on Prologue
EDIT	Erroneous Editing for New Version Update
DATA1	Incomplete or Inconsistent Data Structure Definition
DATA2	Wrong Value for Constant or Preset Data
DATA3	Improper Scaling of Constant or Preset Data
DATA4	Uncoordinated Use of Data by More than One User
DATA5	Erroneous Access or Transfer of Data
DATA6	Erroneous Reformatting or Conversion of Data
DATA7	Improper Masking & Shifting During Data Extraction & Storage
DATA8	Failure to Initialize Counters, Flags, or Data Areas
<u>Miscellaneous</u>	
INTRO	New Error Introduced During Correction
STAND	Noncompliance with Programming Standards and Conventions

was open-ended, that is to say, categories were added as required during the project. The Data category was assigned most frequently (23 percent of total errors), consequently it was divided into eight subcategories. Incomplete or erroneous program design specifications accounted for 15 percent of the total number of errors; logic for 14 percent; and requirements, program design, and access or transfer of data for 10 percent each.

On a similar Army project,<sup>23</sup> Hughes has over a year's experience in using an error classification scheme based on the TRW/RADC scheme. The casual classification was assigned separately from the source phase, and was tailored to the following ten major categories (percent of total errors are shown in parentheses):

- Computational (4)
- Logic (38.5)
- Data definition (20.5)
- Data handling (14)
- Data base (3)
- Interface (4.5)
- Operation (1)
- Documentation (0.5)
- Problem Report Rejection (NA)
- Other (13.5)

The major categories, Data Input and Data Output, were dropped, because they were not appropriate to the application.

An analysis of error trends on this project revealed that eight problems were caused by the improper selection of instructions. Accordingly, it was felt that this class of errors warranted a separate subcategory. Since such a selection could result from either misunderstanding or carelessness, the following two subcategories were added to the Other category:

- Selection of wrong instruction or statement
- Careless selection/omission of instruction or statement

It is believed that these two categories will determine the need for improved training of new programmers on subsequent projects in the understanding of the instruction repertoire. Such categories may be useful in validating complexity metrics such as the one proposed by Ruston.<sup>24</sup> The metric is based on information theory, and assumes that the less frequently an operator or operand is used then the more difficult it is for the programmer to use correctly.

On a Navy project, Hughes employed a code review technique which included the recording of errors according to categories. Five hundred modules had a total of 765 errors; the remaining 742 had no problems.<sup>25</sup> Table VI presents the distribution of the most frequent errors, and compares the distribution with comparable categories from IBM's code inspection technique.<sup>26</sup> The high percentage of errors due to missing or insufficient listing prologues and comments for the Hughes project was probably due to the novelty of such a requirement early in the coding phase.

TABLE VI.—Distribution of errors detected during code inspection

Category	% of Total	
	Hughes	IBM
Prologue/Comments	44	17.0
Design Conflict	19.5	25.5
Logic	11.5	30.5
Programming Standards	11	4.5
Language Usage	5	12.5
Other	3	3.5
Module Interface	3	6.5
Data Base	3	-
Total	100.0	100.0

## RECOMMENDED ERROR CATEGORIES

With respect to proposed error classification schemes, the applicability to more than one project, the excessive granularity and ambiguity of subcategories, have been called out as problems. Hughes has found that the use of a minimal set of three software error classifications (Cause, Severity, and Source) solves these problems and is sufficient to support the assessment of software reliability. As summarized in Table VII, Source tells in which software development phase the error originated in, Cause tells what the analyst or programmer did wrong, and Severity tells whether the manifestation of the error degrades mission performance.

The recommended casual classification for software reliability assessment, containing seven major categories, is shown in Table VIII. The scheme can be tailored by adding subcategories of interest or exception, such as problem rejection, to the Other category. A definition of each category/subcategory is presented in Appendix A.

A severity classification of at least three categories (for example Critical, Major, and Minor) is recommended. In addition to guiding project managers in assigning priorities to the troubleshooting and resolution of problems, severity categories are necessary for practical application of predictive software reliability models. In order for the prediction of residual software faults to be meaningful, the impact of the execution or manifestation of the fault on the system mission performance must also be included. Some proposed reliability models such as the execution time theory model can accommodate severity by running separate predictions for each severity category of interest. The justification for the recommended error casual and source phase categories/subcategories is discussed in the following subparagraphs.

TABLE VII.—A software error classification scheme that provides feedback

<u>Source</u>	- Phase in which error of omission/commission was made (e.g. Requirement, Design, Coding, Test, Maintenance, and Corrective Maintenance).
<u>Cause</u>	- The causal description of the error, rather than symptomatic
<u>Severity</u>	- The resulting effect of the error on mission performance (e.g. Critical, Major, and Minor)

TABLE VIII.—Casual categories to support software reliability analysis

Design	
Nonresponsive to requirements	
Inconsistent or incomplete data base	
Incorrect or incomplete interface	
Incorrect or incomplete program structure	
Extreme conditions neglected	
Interface	
Wrong or nonexistent subroutine called	
Subroutine call arguments not consistent	
Improper use or setting of data base by a routine	
Improper handling of interrupts	
Data Definition	
Data not initialized property	
Incorrect data units or scaling	
Incorrect variable type	
Logic	
Incorrect relational operator	
Logic activities out of sequence	
Wrong variable being checked	
Missing logic or condition tests	
Loop iterated incorrect number of times (including endless loop)	
Duplicate logic	
Data Handling	
Data accessed or stored improperly	
Variable used as a flag or index not set properly	
Bit manipulation done incorrectly	
Incorrect variable type	
Data packing/unpacking error	
Units or data conversion error	
Subscripting error	
Computational	
Incorrect operator/operand in equation	
Sign convention error	
Incorrect/inaccurate equation used	
Precision loss due to mixed mode	
Missing computation	
Rounding or truncation error	
Other	
Not applicable to software reliability analysis	
Not compatible with project standards	
Unacceptable listing prologue/comments	
Code or design inefficient/not necessary	
Operator	
Clerical	

*Category for design-related errors*

Although a casual category for design-related errors is redundant with the source phase category Design, sufficient error volume has been associated with software design activities to warrant a separate casual category. In analyzing designed-related category assignments on three software

TABLE IX.—Misunderstandings as sources of errors during NRL experiment

Category	% of Total Errors
Clerical	36
Design	19
Coding Specs	13
Careless Omission	10
Language	8
Interface	6
Requirements	6
Coding Standards	2
Total	100

projects at Hughes-Fullerton it was found that the categories accounted for 25, 17, and 8 percent of the total errors. Furthermore, the results of the error category frequency distributions collected during code review/inspections (refer to Table V) reveal that design conflicts constitute a significant portion of the error causes (25.5 and 19.5 percent). Another study<sup>8</sup> performed at the Naval Research Laboratory (NRL) reported that design misunderstandings contributed to 19 percent of the total errors (see Table IX).

*Subcategory for clerical errors*

Two experimental studies, one performed at the Naval Post Graduate School (NPGS)<sup>27</sup> and the other performed at the Naval Research Laboratory (NRL),<sup>8</sup> found it necessary to include clerical as a major error category. In fact, both studies found that the clerical category was the most frequency error cause (see Tables IX and X). The NPGS error distributions represent a composite of four projects. On one project the Clerical, Manual subcategory contributed to 36 percent of the total errors. Due to the high occurrences of clerical errors reported on these two unrelated projects, it is recommended that clerical be added as a subcategory to the Other category.

*Maintenance category*

Maintenance errors are defined by Thayer<sup>1</sup> as those errors resulting from the correction of previously documented errors. He reported that in one project this category of errors reached 9 percent of the total number of errors; however, he estimated that a practical norm for this type of error ranges from 2 to 5 percent. Fries<sup>5</sup> reported “. . . a surprisingly high 6.5 percent of the errors were a result of attempts to fix previous errors or update the software. Thus, the number of errors introduced by the correction process itself is nontrivial. This is an important consideration when developing reliability model assumptions.” Note that Fries’ 6.5 percent includes updates or enhancement changes as well as corrections of previously documented errors; therefore the actual percentage value for maintenance errors would probably lie in the 2 to 5 percent range.

TABLE X.—Most frequent error types found during NPGS experiment

Subcategory	% of Total Errors
Clerical, manual	18.5
Coding, Representation	10.0
Coding, Syntax	7.0
Design, Extreme Condition Neglected	6.5
Coding, Inconsistency in Naming	5.0
Coding, Forgotten Statements	5.0
Design, Forgotten Cases or Steps	4.5
Design, Loop Control	4.0
Coding, Missing Declarations or Block	4.0
Coding, Level Problems Limits	3.0
Coding, Sequencing	3.0
Design, Indexing	2.5
Coding, Missing Data Declarations	2.5
Clerical, Mental	2.5
Other (combined)	19.5
Total	100.0

At Hughes-Fullerton three projects have been monitored during development phases for maintenance errors. The portion of total errors for these three projects are 14, 12, and 8 percent. One possible reason these percentages are higher than the previously reported range of two to five percent, is that none of the three Hughes projects controlled the number of allowable patches. Consequently, there was always the extra risk of wrong correction in patch form due to hasty implementation, or the subsequent incorrect symbolic implementation of a successful patch. It is estimated that maintenance errors contribute to as high as 20 percent of the total errors after a system is fielded. Because of the frequency of this type of error, and the interest in reducing the cause of maintenance errors, a separate category is required. Either a Maintenance subcategory could be added to the Other causal category, or a Corrective Maintenance category could be added to the source phase classification. It is recommended that a new category be added to the Source phase classification, because including maintenance error as a causal subcategory would preclude the assignment of the more descriptive cause (e.g., Subscripting error).

#### *Optional category/subcategory assignment*

The original TRW/RADC classification for Project 5<sup>1</sup> was designed for universal application by allowing the option to assign categories at only the major category level (e.g., Computational, Logic, Data Handling, etc.). The TRW study report commented as follows about the applicability of the subcategories: "The detailed categories, however, are less universal and suffer in applicability due to differences in language, development philosophy software type, etc. When data are collected may also have a bearing on applicability [of the detailed categories] to some software test environments. For Project 5 the list used was apparently adequate for the real time applications and simulator software, as well as the Product Assurance tools. However, there was criticism concerning applicability of detailed categories to the real time operating system software problems." Hughes-Fullerton has employed the two-level (category/subcategory) option, and has found it to be satisfactory for all projects.

#### ERROR COLLECTION GUIDELINES

It is human nature not to admit to errors, therefore it is essential that software engineers be informed of the significance of reporting accurate error data to support software reliability analysis. It should be emphasized that the purpose of error reporting is to measure the technology and not the people. I agree with Gerhart's<sup>13</sup> statement: "It is necessary to view errors as a phenomenon of programming which requires study and, while it is necessary to be sensitive to peoples' reactions when threatened by exposure of errors, it may be healthier to get the errors and the errants out in the open rather than to cover up the human origin of errors."

Automatic data collection may be the only means to ensure

objective data, but short term projects cannot afford it. In most instances, useful software reliability information can be obtained by only slight modifications to existing problem report/correction systems. The use of coded error category descriptors on program trouble and correction reports tends to alleviate thoughts of incrimination.

Guideline procedures for assigning and approving error categories should be included in project standard practices to promote consistent interpretation of the error categories. In addition to the error categories, the procedure should contain detailed definitions of the error subcategories. Those definitions guide individual programmers in assigning the most appropriate category to represent the error at hand. Even with the use of such an error category dictionary, programmers may assign different categories for the same errors. Therefore, it is further suggested that a senior programmer or reliability analyst be responsible for reviewing all error category assignments for consistency and accuracy. Certain less offensive subcategories such as Clerical require special monitoring, because a programmer will lean toward them when given a choice.

Programmers must be reminded to fill out a separate problem correction report for each distinguishable correction at the module level. It is recommended that the following data be collected in addition to the error classifications:

- Date/time that error/incident was detected
- Date/time that error was resolved by programmer
- Date/time that resolution was verified
- Principle module responsible for error

#### CONCLUSIONS

It appears from the survey of proposed software error classification schemes that they differ primarily because of varying emphasis on different areas of software development. I agree with some researchers that error classifications must reflect areas of interest, however this does not preclude the development of a standard minimal set of software error classifications that has universal application—including reliability assessment. Therefore, I suggest that the proposed error classification scheme be considered as a standard for use in software reliability assessment. The proposed scheme can be used during design reviews, code reviews, and testing.

In order to satisfy all activities, additional error characteristics will have to be collected. For example, in the validation and use of predictive software reliability models the date and time of detection of a fault, and the date and time of correction of the error are additional data that are required to be collected. However, if the cause of the "error" is ignored a reliability model could be fed time/date data for a problem report, such as integration of new software, that is not analogous to the residual class of errors that quantitative models predict.

The development of a set of standard software error classifications is a prerequisite for the development of a mean-

ingful software reliability discipline. Such a set of classifications can serve two promising approaches to the discipline: 1) those that emphasize the use and assessment of reliability-producing techniques during the early development phases, and 2) those that focus on the prediction and measurement of the number of residual errors after acceptance, by statistical math models. Both approaches require error classifications to effectively assess and measure software reliability.

Concurrent with the development and acceptance by the software community of a standard set of causal, severity, and source classifications there is a need for research and development in the automation error collection through compilers and test runs. Also, the capabilities of emerging independent V&V tools when augmented by standard error classifications can be extended to improve test plan and procedure generation.

## REFERENCES

1. Thayer, T. A., et al, "Software Reliability Study," TRW-Redondo Beach, RADC TR-76-238 (Aug 1976).
2. Willmorth, N. E., "Proceedings of Data Collection Problem Conference," RADC TR-76-329, Vol. VI (Dec 1976).
3. Finfer, M. C., "Software Data Collection Study," System Development Corp., RADC-TR-76-329, Vol III (Dec 1976).
4. Baker, W. F., "Software Data Collection and Analysis: A Real-Time System Project History," IBM Corp., RADC-TR-77-192 (Jun 1977).
5. Fries, M. J., "Software Error Data Acquisition," Boeing-Seattle, RADC-TR-77-130 (April 1977).
6. Chief of Naval Materiel, Military Standard for Weapon System Software Development MIL-STD-1679 (Navy), AMSC No. 23033 (Dec 1978).
7. Willmorth, N. E., et al, "Software Data Collection Study, Summary and Conclusions," RADC-TR-76-329, Vol. I (Dec 1976).
8. Weiss, D. M., "Evaluating Software Development by Error Analysis: The Data from the Architecture Research Facility," Naval Research Laboratory, NRL report 8268 (Dec 1978).
9. Nelson, R., "Software Data Collection and Analysis, Draft"—partial report, RADC (Sep 1978).
10. Gannon, C., "Error Detection Using Path Testing and Static Analysis," *Computer*, pp 26-31 (Aug 1979).
11. Hecht, H., "Measurement, Estimation, and Prediction of Software Reliability," Aerospace Corp. NASA CR-145135 (Jan 1977).
12. Motley, R. W. and Brooks, W. D., "Statistical Prediction of Programming Errors," IBM Corp. RADC TR-77-175 (May 1977).
13. Gerhart, S. L., "Development of a Methodology for Classifying Software Errors," Duke University (July 1976).
14. Castle, S. G., "Software Reliability: Modelling Time-to-Error and Time-to-Fix," masters thesis, Air Force Institute of Technology (Mar 1978).
15. Kruszewski, G., "Modeling Software Reliability Growth, Proceedings of Surface Warfare Systems RMQ Seminar," Norfolk, VA (Sept 1978).
16. Schafer, R. E., et al, "Validation of Software Reliability Models," Hughes-Fullerton, RADC-TR-79-147 (Aug 1979).
17. Sukert, A., "State of the Art in Software Reliability," Presentation, NSIA Software Conference, Buena Park, CA (Feb 1979).
18. Thibodeau, R., "The State-of-the-Art in Software Error Data Collection and Analysis," AIRMICS (Jan 1979).
19. Amory, W. and Clapp, J. A., "Engineering of Quality Software Systems (A Software Error Classification Methodology)," MITRE Corp., MTR-2648, Vol VII, Jan 1975, also RADC-TR-74-324, Vol VII.
20. Rubey, R. J., "Quantitative Aspects of Software Validation," *Proceedings of the 1975 International Conference on Reliable Software* Los Angeles, pp 246-251 (April 1975).
21. NAVSEA, Statement of Work for AN/SLQ-32(V) Verification and Validation, Appendix A (May 1977).
22. Hartwick, R. Dean, "Software Acceptance Criteria Panel Report," Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resource Management, Software Workshop, Monterey, CA (April 1979).
23. Bowen, J. B., "AN/TPQ-36 Software Reliability Status Report," Hughes-Fullerton, CDRL 8-18-015 (Dec 1979).
24. Shooman, M. L. and Ruston, H., "Summary of Technical Progress, Investigation of Software Models," Polytechnic Institute of New York, RADC-TR-79-188 (July 1979).
25. Thielen, B. J., "SURTASS Code Review Statistics," Hughes-Fullerton, IDC 78/1720.1004 (Jan 1978).
26. Fagan, M. E., "Inspecting Software Design and Code," *Datamation*, pp 133-144 (Oct 1977).
27. Hoffman, H., "An Experiment in Software Error Occurrence and Detection," masters thesis, Naval Postgraduate School (Jun 1977).

## APPENDIX A

### DEFINITION OF RECOMMENDED ERROR CATEGORIES/SUBCATEGORIES

#### *Design*

The Design category reflects software errors caused by improper translation of requirements into design. The design at all levels of program and data structure is included (subsystem through module and data base through table). Such errors normally occur in the design phase, but are not limited to that phase. Errors due to inconsistent, incomplete, or incorrect requirements do not qualify for this category; such errors should be assigned to the subcategory, "Not Applicable to Software Reliability Analysis."

#### *Interface*

The Interface category includes those errors concerned with communicating between 1) routines and subroutines, 2) routines and functions, 3) routines and the data base, 4) the executive routine and other routines, and 5) external interrupts and the executive routine.

#### *Data definition*

This category pertains to errors involved with permanent data, such as retained, global, and COMPOOL. It includes common variable and constant data, as well as preset, initialized, and dynamically set variables.

#### *Logic*

The Logic category includes all logical-related errors at the intramodule level. Examples of this category are incorrect relational operators and incorrect looping control. Improper or incomplete logic occurrences at the intermodule level do not qualify for this category, and should be assigned to the Interface category.

*Data handling*

The Data Handling category is concerned with errors in the initialization, accessing, and storage of local data; as well as the conversion and modification of all data.

*Computational*

The Computational category pertains to inaccuracies and mistakes in the implementation of addition, subtraction, multiplication, and division operations.

*Other*

The Other category is designed to provide flexibility for each application. However once selected for a project, the

subcategories should not change. The following suggested subcategories deserve further explanation.

*Operator*

This subcategory includes errors caused by inaccurate users manuals for both operational and diagnostic applications.

*Clerical*

This subcategory includes errors that can be traced to careless keypunch, configuration control, or system generation operations.



# What makes a reliable program—Few bugs, or a small failure rate?\*

by B. LITTLEWOOD

*Mathematics Department  
The City University  
London EC1V OHB  
England.*

## INTRODUCTION

It is instructive to look at some of the reasons advanced by software developers for their reluctance to use software reliability measurement tools. Here are a few common ones:

(A) "Software reliability models are statistical. Programs are deterministic. If certain input conditions cause a malfunction today, then the same conditions are certain to cause a malfunction if they occur tomorrow. Where is the randomness?"

(B) "I am paid to write reliable programs. I use the best programming methodology to achieve this. Software reliability estimation procedures would not help me to improve the reliability of my programs."

(C) "We verify our software. When it leaves us it is correct."

(D) "I ran your software reliability measurement program on some data from a current project of ours. It said there was an infinite number of bugs left in the program. Who are you trying to kid?"

(E) (same manager as in D, but one week later) "We corrected a couple of bugs and ran the reliability measurement program again. This time it said that there were 200 bugs left. Infinity minus two equals two hundred? Is this the new math?"

(F) "We put a lot of effort into testing. The selection of test data is a systematic process designed to seek out bugs. Reliability estimation based on such test data would be no guide to the performance of the program in a use environment."

(G) "We are writing an air traffic control program. Total system crash would be catastrophic. Other failures range from serious to trivial. Reliability models do not distinguish between failures of differing severity."

Although I have been involved in software reliability modelling for the past decade, and have myself perpetrated a few models, I have a great deal of sympathy with some of

the sentiments expressed above. I have a feeling that some of the early models have been oversold, that not enough emphasis has been placed on the underlying modelling assumptions, and that by concentrating on a simple reliability analysis we might be ignoring wider concerns. In this paper I shall be looking at one common deficiency of early models and suggesting a way in which it can be overcome. I hope that, in passing, some new insight into the wider issues will be gained.

## THE PROBLEM AND ITS EARLY SOLUTION

In its simplest form the problem is this. We have available some data  $t_1, t_2, \dots, t_n$ , representing successive (execution) times between failures of a program. What can we say about the current reliability of the program, and how this will change in the future?

This bald description needs some amplification. In the first place, are we sure what we mean by "reliability" in this context? In A, above, we see one of the difficulties. There is a sense in which software failures are completely predictable: if we know that an input caused a failure in the past, then the same input will cause a failure now (assuming the program is unchanged). Equally, if a program can correctly process an input once, the same program can correctly process the same input forever. Contrast this situation to that of hardware, from which conventional reliability terminology arises. Hardware devices exhibit wear-out and it is not possible to guarantee that the response of a device to a particular input will remain constant. More strongly, we can say that a hardware device is certain to fail ultimately, whereas a program, if perfect, is certain to remain failure-free. Of course, it is questionable whether there is much chance of writing a real-life program in such a way that it is perfect. The principle, however, remains: it is possible to conceive of a program which is never going to fail. This concept of the "perfect" program immediately suggests a way to define software reliability which would not have a hardware parallel. A program which will never fail is one

\* This research was supported in part by the US Army, European Research Office, under Grant No. DAERO-79-G-0038.



containing no "defects": no errors (or bugs). The "reliability" of a program is its relative freedom from bugs. Such a concept of reliability, then, is essentially static: it describes the state of the program rather than how the program performs (its failure rate, mean time to failure, etc.). My own view is that we are almost always more concerned with the dynamic reliability of a program than the number of bugs it contains. There are, though, some situations where the number of bugs remaining in a program is of practical interest: the commonest such situation being that where we wish to be assured that *none* are left. It seems sensible, therefore, that we should have reliability models which enable both of the following interpretations of reliability to be used: relative freedom from bugs, relative freedom from failures in operation. It is the relationship between these two concepts of reliability—how the number of bugs remaining in a program affects the performance of the program—which will form the main theme of this paper.

This seems a convenient place to comment on C. When I talk of a perfect program I mean something *more* than correctness. There seem to be two basic objections to formal verification of programs. Most important is the logical objection: the most that can be achieved is a proof that the program is consistent with its specification, not with the informal requirements [1]. Those advocates of program verification who maintain that a program can be "correct," and yet fail to fulfill the requirements demanded by the customer, are just passing the buck. A problem does not disappear by declaring it to be someone else's responsibility. Another objection, which may ultimately be overcome, is that of cost: the sheer effort required to verify programs of realistic size is often completely prohibitive. This seems likely to remain true for a long time. I do not mean to imply that these ideas are not valuable, though. On the contrary, it is clear that they have already had a quite far-reaching and valuable impact on programming methodology.

Notice, by the way, that a program could perform "perfectly" and yet fail a proof of "correctness." Although we would be right to reject the program if we knew the result of the proof, it is clear that in the absence of such knowledge it may be possible to describe the program as highly reliable.

When we look at large real-life programs, written under time and cost constraints, discussions about perfectibility seem merely theological. We shall be almost certain that such programs do contain bugs, that they will eventually produce unacceptable output, and that proofs of correctness (if they were feasible) would fail. Our purpose, then, is to quantify this imperfection: this is why we need reliability studies.

Returning now to the basic problem, it is important to be aware of the source of the inter-failure times  $t_1, \dots, t_n$ . In most cases this data is collected during the test and integration phase of the project, whilst debugging is in process. We would expect, then, that the reliability of the program is increasing: i.e., there will be a tendency for the  $t$ 's to be increasing. At any particular stage of this process it is the intention to use the model to measure the current reliability and *predict future reliability*. This brings us to F. These

models can predict future performance of a program only on the assumption that there is continuity in the behaviour of the programming team and in the behaviour of the process selecting inputs. This assumption is commonly violated, and in such cases model predictions cannot be trusted. Possibly the commonest situation of this kind is when there is a discontinuity between the test and use environments. In many cases it is simply impractical, or prohibitively slow and expensive, to use an actual (or simulated) use environment to produce inputs for the test phase. Instead, inputs are generated with the specific intention of testing most rigorously those parts of the program which are known a priori to be likely to contain errors: a similar process is often used for hardware, called *stress testing*. It may sometimes be possible to use data from *other* projects to estimate the relationship between the severities of the test and use environments—what Musa [2] calls the *testing compression factor*. My own view is that this will rarely be justified, since every program is essentially unique. We would need to know not merely that the inputs for test and use environments were related similarly between the current program and its predecessors, but that *responses* of the programs to these inputs had the same relationship.

The other source of discontinuities of behaviour which prevent direct use of these models is system integration. If new modules are being integrated into the system during the collection of the  $t$ 's, then new sources of failure are being introduced and it is unreasonable to expect the estimates based upon an earlier stage of system integration to be valid. It does seem likely in this case, though, that estimation of the magnitude of the reliability discontinuities will sometimes be possible. There is likely to be greater commonality of behaviour between modules of the same system than between different projects. This is an area where further research is needed; at present we shall have to assume that the models are used only after integration, or for the periods of homogeneous behaviour between module integrations.

It may seem, after these areas have been eliminated, that there is very little that software reliability models can be used for. However, if we have a system which has been totally integrated, and we are sure that the test environment (simulated or real) is representative of the use environment, we can use the models to estimate current reliability and predict future reliability *during debugging*. In those cases where it is possible to test *modules* under these conditions, then of course the same reliability estimation can be performed on them. It may even be possible to combine knowledge of the reliability of the modules with structural information about their roles in the system and calculate overall system reliability [3, 4].

Let us now return to the general problem and look at the early solutions. I shall use the notation of Jelinski and Moranda [5], but the models of Shooman [6] and Musa [2] are essentially the same (although it should be noted that Musa introduces many extra refinements over the basic model). It is assumed that the random variables  $T_i$ , representing the times between  $(i-1)$ th and  $i$ th failures of the program are *independent* and have the exponential distributions:

$$pdf(t_i) = \lambda_i e^{-\lambda_i t_i}, \quad t_i > 0 \quad (1)$$

where  $\{\lambda_i: \lambda_i > 0\}$  is the sequence of *failure rates* of the program. Note that Musa argues cogently for "time" in this context to represent *execution time*, rather than calendar time.

The reasoning behind assumption (1) is that the input space contains a subset of inputs which will induce failure and that this subset is *encountered randomly*. The process can thus be viewed as a Poisson process with a rate which changes at each event. The assumption seems to be a reasonable one so long as we define "failure" fairly carefully. We would, for example, have to treat a cascade of failures, caused by a single error in the program being encountered once, as a single failure. This accords with usual practice.

The important remaining question concerns the structure of the sequence  $\{\lambda_i\}$ . It is clearly impossible to estimate each  $\lambda_i$  separately, since there will generally only be a single observation,  $t_i$ . More importantly, we wish to be able to project  $\lambda_i$  for *future*  $i$ . Jelinski and Moranda make the following assumption (similar assumptions can be found in [2,6]:

"The failure rate at any time is assumed proportional to the current error content of the program . . . the proportionality constant is denoted by  $\phi$ . . . ." ([5], p. 473).

This is equivalent to assuming

$$\lambda_i = (N - i + 1)\phi \quad (2)$$

where  $N$  is the number of bugs (errors) in the program before debugging starts. Each remaining bug contributes an amount  $\phi$  to the overall failure rate of the program, so that when  $(i - 1)$  bugs have been eliminated there remain  $(N - i + 1)$ . Of course, this assumes that each *failure* of the program results in the immediate removal of one *bug*. In fact it is relatively easy to relax this assumption in order to represent imperfect debugging; this is an issue which I shall not examine here.

The model is now completely specified by the two unknown parameters  $N$  and  $\phi$ . These can be estimated from the data  $t_1, t_2, \dots, t_n$  by, say Maximum Likelihood, and estimates of current and future reliability calculated.

## A NEW SOLUTION

Consider the quotation from [5] which results in (2). What is being assumed is that each bug in the program contributes equally to the overall failure rate of the program. Thus when a failure occurs (and a bug is fixed) the overall failure rate drops by a fixed constant amount,  $\phi$ . Between bug-fixes the failure rate remains constant. A plot of failure-rate against execution time is shown in Figure 1: all steps are of equal size.

It seems to me wrong to assume all bugs have the same effect on the overall failure rate. In fact it seems likely that the contributions from different bugs to the failure rate of the program will vary quite widely. There is, for example, evidence that the frequencies with which different portions

of code are exercised vary enormously. A bug in frequently exercised code will cause failures more frequently than a bug in infrequently exercised code (other factors being equal), i.e., it will contribute more to the failure rate of the program.

A more plausible scenario, then, is that at the beginning of debugging the program contains a pool of  $N$  bugs with differing failure rates. Early failures of the program are more likely to be caused by those bugs with the greatest failure rates. Thus early bug-fixes, corresponding to the removal of bugs with larger failure rates, will have greater effect on the overall failure rate. Instead of a plot such as Figure 1, the steps will be of different sizes with larger steps occurring early in the debugging.

Before suggesting in detail how this effect can be modelled, it is instructive to examine the source of the random variation in software failure times as suggested by earlier authors [2,5,6]. All these models assume that the *sole* source of randomness (or uncertainty) lies in the nature of the input stream. Thus in (1),  $\lambda_i$  is treated as a *constant* (given by (2) if  $N, \phi$  known) and the only random variable is  $T_i$ . This seems to me to ignore the uncertain nature of program writing and debugging itself. Since we shall be uncertain of the amount any bug contributes to the overall failure rate, we are uncertain of the relationship between  $\lambda_{i-1}$  and  $\lambda_i$ . Thus instead of a sequence  $\{\lambda_i\}$  with a *deterministic* relationship between successive terms, as in (2), we should be dealing with a sequence  $\{\Lambda_i\}$  of *random variable* failure rates. Another way of looking at this is as follows. Instead of treating the debugging process as a series of deterministic operations on "a program," it can be viewed as the creation of a series of programs,  $P_1, P_2, \dots, P_n$ . Program  $P_i$  may differ from program  $P_{i-1}$  in only a small way—the result of fixing the  $(i - 1)$ th bug—but *it is a different program*, and the difference is *unpredictable*. Just as it is not possible to predict what the sequence of debugging changes will be which produce the sequence  $\{P_i\}$  of programs, so the sequence of failure rates is not predictable. It must be treated as a sequence of random variables  $\{\Lambda_i\}$ .

The two sources of uncertainty can be modelled in the following way. Assume, as do earlier authors [2,5,6], that the uncertainty in the input stream causes the execution time to next failure to be conditionally exponentially distributed. That is,

$$pdf(t|\Lambda = \lambda) = \lambda e^{-\lambda t}, \quad t > 0. \quad (3)$$

We shall assume perfect debugging, for simplicity. So that when  $i$  failures have occurred we have removed  $i$  bugs. Let total execution time be  $t^{(0)}$  at this stage (see Figure 3). Then

$$\Lambda = \Phi_1 + \Phi_2 + \dots + \Phi_{N-i} \quad (4)$$

where  $N$  is the initial number of bugs (unknown) and  $\Phi_r$  represents the (random variable) contribution to the overall failure rate of the  $r$ th bug among the remaining  $(N - i)$  bugs. It solely remains to find the distribution of  $\Phi_r$  for all  $t^{(0)}, r$ . Clearly

$$pdf(\phi_r) \equiv pdf(\phi_s) \text{ for all } r, s.$$

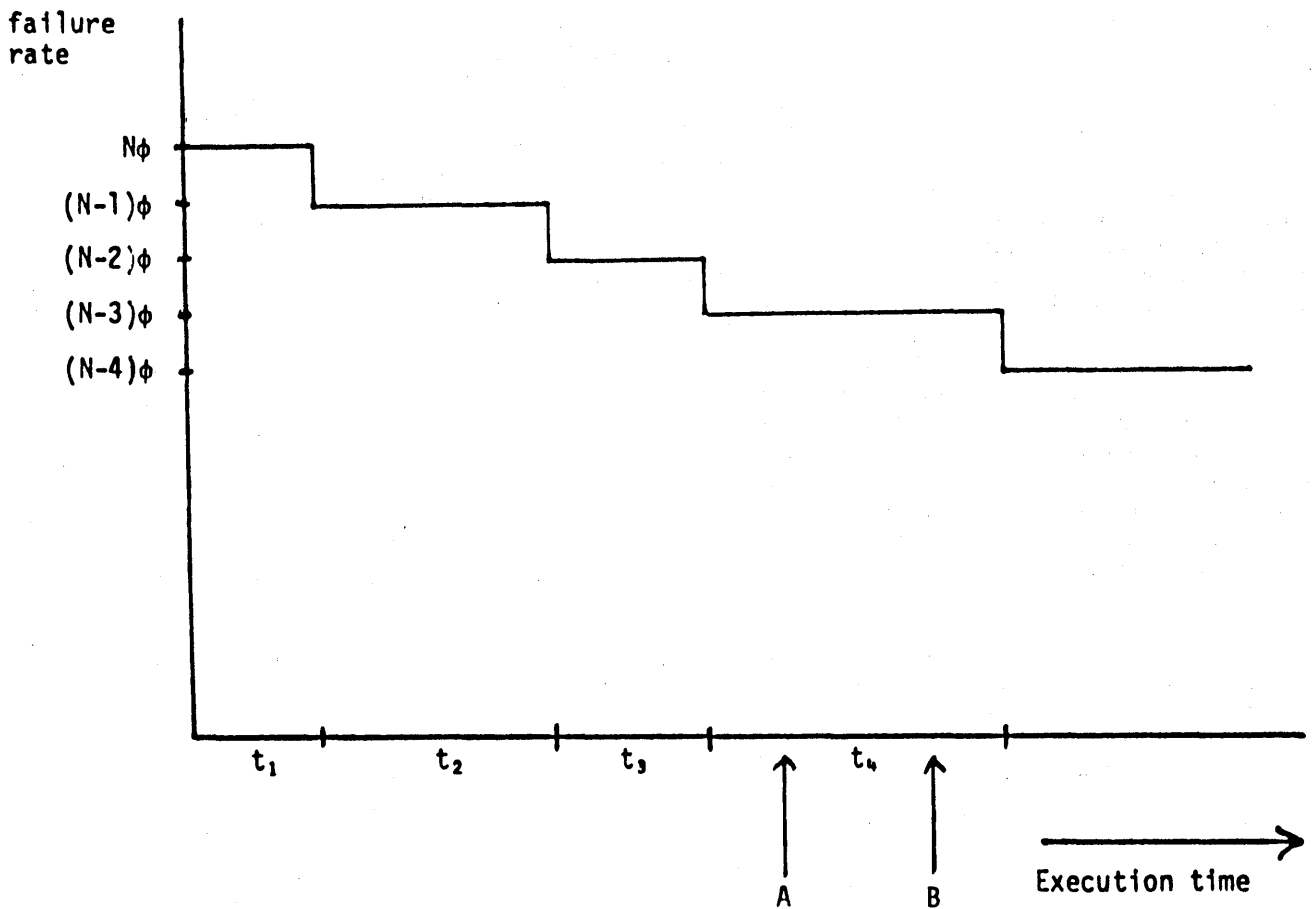


Figure 1.

This merely states that at each stage our uncertainty (our ignorance) about the bugs which remain is the same for each: we cannot distinguish between them.

Let us represent our initial uncertainty about the  $\Phi$ 's by a Gamma ( $\alpha, \beta$ ) distribution:

$$pdf(\phi) = \frac{\beta^\alpha \phi^{\alpha-1} e^{-\beta\phi}}{\Gamma(\alpha)} \quad (\phi > 0). \quad (5)$$

Then the distribution of each of the  $\Phi$ 's in (4), by Bayes Theorem, is

$$pdf(\phi | \text{bug has survived detection for a time } t^{(0)}) \quad (6)$$

$$= \frac{Pr\{\text{no failure of this bug in } (0, t^{(0)}) | \Phi = \phi\} \cdot pdf(\phi)}{\int Pr\{\text{no failure of this bug in } (0, t^{(0)}) | \Phi = \phi\} \cdot pdf(\phi) d\phi}$$

Substituting (5) into (6) and simplifying we find that the distribution of a  $\Phi$  in (4) is

$$\text{Gamma}(\alpha, \beta + t^{(0)}). \quad (7)$$

Since the sum of independent, identically distributed Gamma random variables is itself Gamma distributed, we find from (4) that the distribution of  $\Lambda$  is

$$\text{Gamma}((N-i)\alpha, \beta + t^{(0)}) \quad (8)$$

Finally, from (3) and (8) we find that the distribution of the time to next failure,  $T$ , when  $i$  bugs have been detected and execution time  $t^{(0)}$  has elapsed (Figure 3) is

$$pdf(t) = \int_0^\infty pdf(t | \Lambda = \lambda) pdf(\lambda) d\lambda \quad (9)$$

$$= \frac{(N-i)\alpha(\beta + t^{(0)})^{(N-i)\alpha}}{(\beta + t^{(0)} + t)^{(N-i)\alpha + 1}},$$

which is a *Pareto distribution*. This result should be compared with the exponential distribution of the Jelinski-Moranda model, (1) and (2). Full details of this new model can be found in [7], including examples of how it can be used to predict future reliability.

Consider the failure rate at the arrowed epoch in Figure 3. This is

$$\frac{(N-i)\alpha}{\beta + t^{(0)}}. \quad (10)$$

Notice how this changes as debugging proceeds. When a failure occurs and a bug is fixed, the failure rate drops by an amount  $\alpha/(\beta + t^{(0)})$ ; early bug-fixes, with small  $t^{(0)}$ , cause greater reductions in the program's failure rate than later

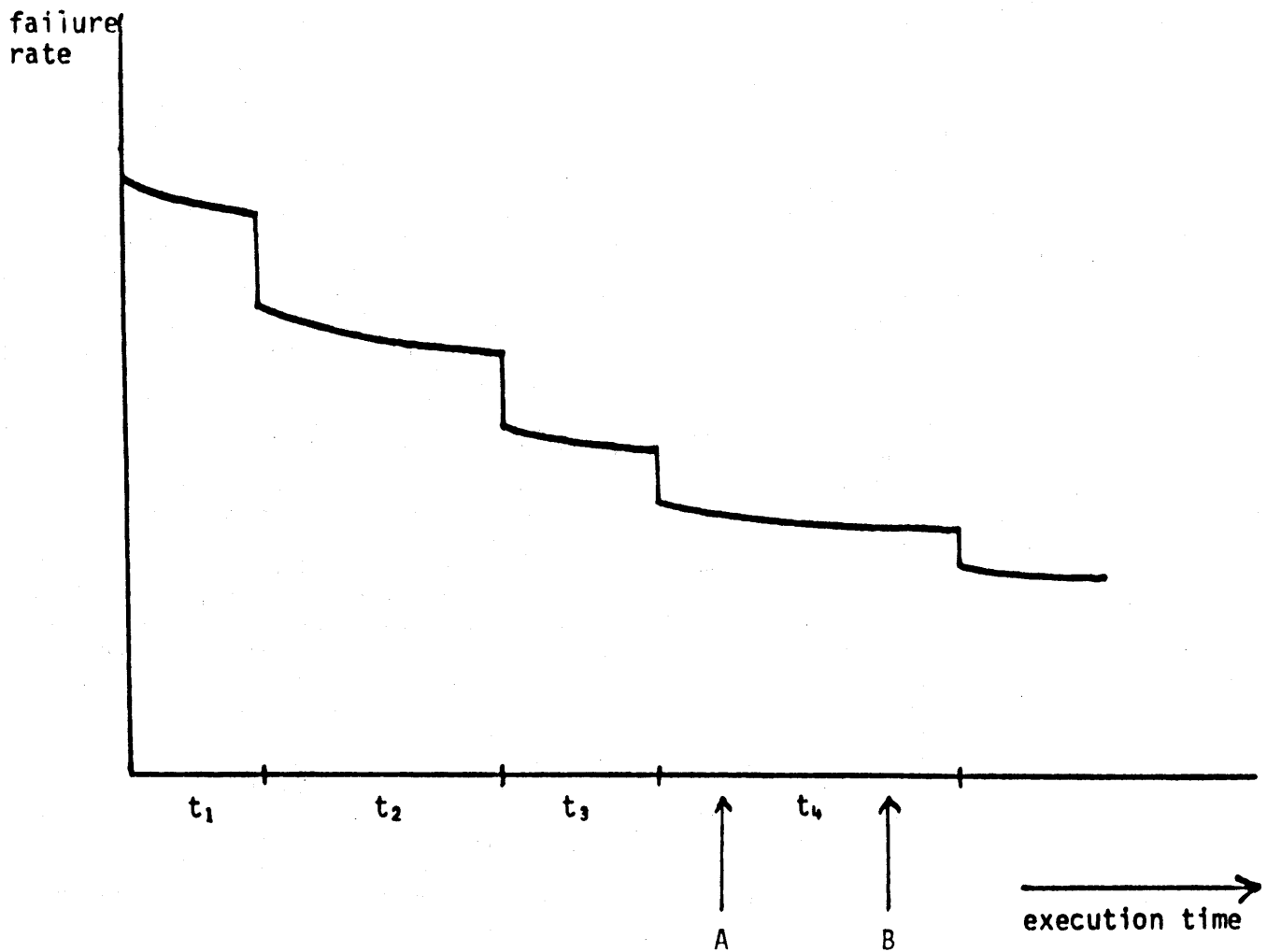


Figure 2.

ones. During periods of failure-free operation, between bug-fixes, the failure rate decreases continuously as  $t^{(0)}$  increases. We thus get a plot of failure rate against time of the kind shown in Figure 2.

As a justification of the *decreasing failure rate* (DFR) property of the Pareto distributions, consider the two epochs A and B in Figures 1, 2. Assume that a judgment of the reliability of the program has been made at A. How would you expect the reliability to have changed at B, after a further period of failure-free operation? It seems to me more plausible that we should be reassured by the extra evidence (Figure 2), since this is evidence of good performance, than

that we should believe the reliability unchanged (Figure 1). What is in fact happening, in this model, between A and B is that we are gathering new information about the distribution of the failure rates of remaining bugs—specifically we are increasing  $t^{(0)}$  in (7).

IMPLICATIONS OF THE NEW MODEL

The intention behind all models of this kind is the same. We wish to be able to estimate both static reliability (number of remaining bugs) and dynamic reliability (frequency of failures) of a program. I have argued in the previous section that early models make a false assumption about the relationship between these two measures. Let us now look at the implications of the new model for reliability estimation and, in particular, what consequences would follow from using one of the naive models.

It should be acknowledged, first, that the new model is a little more complicated. It is necessary to estimate three parameters ( $N, \alpha, \beta$ ) from the available data, rather than the

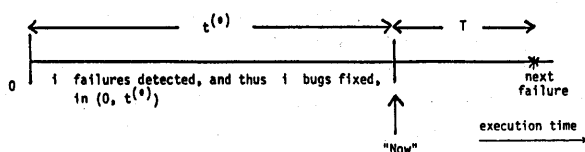


Figure 3.

two parameters ( $N$ ,  $\phi$ ) of the earlier models. This should not present any unusual difficulties.

An important observation is that the Jelinski-Moranda model is a special case of the new model. If we let  $\alpha \rightarrow \infty$ ,  $\beta \rightarrow \infty$  in such a way that  $\alpha/\beta = \phi$  in (5), we find that the Gamma ( $\alpha, \beta$ ) distribution becomes concentrated at  $\phi$ . Thus, if a particular data set produces values of  $\alpha$  and  $\beta$  which are very large, the Jelinski-Moranda model will provide a good approximation with  $\phi = \alpha/\beta$ . Of course, this operates in reverse: if the best-fitting values of  $\alpha$  and  $\beta$  are not large, this implies that the Jelinski-Moranda model would be a poor approximation to the underlying process. In summary, then, nothing can be lost by using the new model instead of the old ones; and something important may be gained.

Assuming that the model does not reduce to the Jelinski-Moranda one, in what ways will it give a different picture of the reliability growth taking place during debugging?

In the first place, it suggests that there is a law of diminishing returns operating in debugging. The reliability improvements gained from successive bug-fixes gradually become smaller and smaller. This implies that estimates of  $N$  may be larger than for the Jelinski-Moranda model without necessarily implying equivalently large estimates of dynamic reliability (e.g. failure-rate). This law of diminishing returns suggests that it will often be appropriate to end debugging before the program is judged bug-free, without implying that such a program is unreliable. In other words, if we are solely interested in the performance of the program (failure rate, mean time to failure, etc.) we can accept a program known to contain many bugs, as long as we are assured that these bugs cause failures infrequently. This seems to me to accord better with intuition and real-life practice than the Jelinski-Moranda assumption, which deems all bugs to have the same contribution to overall reliability. We have all, I think, encountered programs containing bugs which we were prepared to live with.

My own view, then is that almost always the appropriate criterion to adopt is dynamic reliability rather than number of remaining bugs. There are, though, situations where we might wish to have a very high assurance that no bugs remained. Examples would be an air traffic control system or nuclear power station safety system. It would not be sufficient to know that the program was very reliable, whilst containing bugs, if these bugs included ones with catastrophic consequences. This observation reveals the weakness of an analysis purely in terms of the *counting* of failures and bugs (see quotation G). What we ought to have are models which enable us to predict the process of *consequences* of failures. There is, unfortunately, little data or research in this area—no doubt partly due to a natural reluctance to accept a quantification of the unthinkable. We demand, instead, a high assurance that the system is “perfect.”

If we wish to stop debugging only when we have a high assurance that the last bug has been removed, the new model gives some disturbing answers. Since the model will often tend to suggest that many bugs remain (albeit ones with small failure rates), and the successive times between their re-

movals are Pareto distributed (the Pareto distribution has a long tail, i.e. large values occur with greater frequency than in the exponential case), we find that estimates of the time required to end debugging are very large. Often, with large systems, they will be prohibitively large. It is, effectively, *impossible* to make a large system bug-free.

Again, this is not surprising: it seems to accord with experience. But it is worrying. Contrast this with the hardware case: one of the important results of hardware reliability is that it is possible to make a system with any given reliability using components of any given unreliability. We cannot do this for software. Does this mean that we cannot use software for such critical applications? In practice we seem to have little choice.

## SUMMARY AND CONCLUSIONS

The new model that I have described does, I think, represent the relationship between static and dynamic measures of software reliability more naturally than earlier models. I would not, however, suggest that this or any other model is definitive. Indeed, I suspect that it will be a long time before we are able to apply these techniques with confidence to every software development project. In the meantime, we have some techniques which are useful when treated with care: in particular, it is necessary to be sure that the underlying modelling assumptions do apply to the project under examination. So my reply to speaker B would be that, whilst agreeing that software reliability techniques do not of themselves help to improve reliability, it is a brave manager who asserts that his programs are reliable without in some way measuring this reliability.

On the debit side, there is still a great deal of work remaining. In my view, the single biggest gap in our knowledge lies in the area of costs/consequences of failures. We have little in the way of theory, and very little data; yet we all recognise that a reliability theory is only one step on the road to a more comprehensive cost theory. This is an area which urgently requires study.

Another area where progress has been disappointing is that of structural models. It seems intuitively clear that the structure of a program will affect its reliability, but so far there is no effective way of incorporating into a reliability model the wealth of information available about program structure.

Finally, a comment on quotations D and E. It is true that “difficulties” have been experienced with parameter estimation of the early models, and this has tended to alienate some potential users. It should be said that these problems generally only occur with small data sets (i.e. at the very beginning of the debugging period), when the evidence for growth in reliability is slight. Since all the models depend upon an assumption of reliability growth, it is not surprising that things can go wrong when such growth is not clearly evident in the data. It must always be borne in mind that these techniques are not a magical panacea: they are simply systematic methods of estimating what is actually present in data.

## REFERENCES

1. DeMillo, R. A., Lipton, R. J. and Perlis, A. J., "Social processes and proofs of theorems and programs," *Comm. ACM* May 1979, Vol. 22, No. 5, pp. 271-280.
2. Musa, J. D., "A theory of software reliability and its application," *IEEE Trans. on Software Engineering*, Vol. SE-1, Sept. 1975, pp. 312-327.
3. Littlewood, B., "A reliability model for systems with Markov structure," *Applied Statistics (J. Royal Statist. Soc., Series C)*, Vol. 24, No. 2, 1975, pp. 172-177.
4. Littlewood, B., "A software reliability model for modular program structure," *IEEE Trans. on Reliability* (Special Issue on Software Reliability), Vol. R-28, No. 3, August 1979, pp. 241-246.
5. Jelinski, Z. and Moranda, P. B., "Software reliability research," in *Statistical Computer Performance Evaluation*, Ed.: W. Freiberger. New York: Academic, pp. 465-484.
6. Shooman, M., "Operational testing and software reliability during program development," *Record 1973 IEEE Symposium on Computer Software Reliability*, New York, NY, April 30-May 2, 1973, pp. 51-57.
7. Littlewood, B., "A Bayesian differential debugging model for software reliability," in *Proceedings of Workshop on Quantitative Software Models*, Kiamesha Lake, NY, Oct. 9-11, 1979 (to appear).



# Software reliability and advanced avionics

by GERARD E. MIGNEAULT

NASA Langley Research Center  
Hampton, Virginia

## SUMMARY

This paper proposes that software is becoming the most safety critical element of the highly reliable avionics systems which will be needed in civil transport aircraft of the future.

The paper first discusses the pressures leading to the use of digital technology, especially computers with software, in future civil transport aircraft. The level of required reliability pertaining to safety is then determined, both as mandated by regulations and as observed in actual practice.

Finally, advanced fault tolerant computers are described. Their reliability is simply analyzed in order to determine the role software will play; it is critical. The level of software reliability required is then examined.

## INTRODUCTION

Electronic components are not new to civil transport aircraft. However, in the past such equipment has been predominantly analog in nature. The recent past, the present and the near future constitute a period of transition, a time of change from analog components to components making fuller use of digital technology, and including the use of stored program computers.<sup>1</sup> One can foresee the trend continuing into the future and leading to greater dependence upon stored program computers, and consequently, upon the embedded software. It follows therefore, that software reliability is becoming, and will continue to become, an even more significant factor in the analysis of the reliability of avionics and, accordingly, of the reliability of the total aircraft.

## AVIONICS IN FUTURE CIVIL AIRCRAFT

The anticipated wider use of stored program computers in future civil aircraft is a consequence of several more or less obvious factors. More obvious is the pace of development of digital technology: ever greater amounts of computational capability compressed into ever smaller volumes—of less weight, consuming less power, and at less cost both initially and in later maintenance. Thus, other things unchanging, there is an economic advantage to the use of the newer technology devices—on a functional device

per functional device substitution basis. Since the newer, stored program computer devices are essentially multifunction devices, there is also the potential benefit of reduction of the total number of devices and greater standardization of device types. Moreover, multifunction devices permit priority rankings among functions to be exercised, i.e., the option is more available to the system of choosing which functions continue to perform when the system is faced with a number of failed components. Such an option is no insignificant advantage; it is not available to a system in which each component device is dedicated to a specific function. Thus, multifunction devices provide a system with a greater likelihood of "graceful degradation." Another more obvious factor is the opportunity for expanding the scope of some functions by virtue of the increased data base and computational power available.

A less obvious factor is the fact that some potential increases in civil aircraft fuel efficiency are dependent upon the availability of reliable, increased computational capability. Such fuel efficiency increases would be possible due to weight and drag savings resulting from reductions in aircraft passive structure. In turn, the structure reductions would become possible by means of "active control" techniques which maintain aircraft aerodynamic stability and reduce peak local loads on aircraft structure (in effect, distribute loads more evenly across the structure) by complex, precise, ever continuing, and possibly differential, "active control" of aircraft control surfaces. Clearly, to the extent to which such computational power was substituted for previously passively provided structural integrity and stability, uncontrolled in-flight interruptions and/or continuing general malfunctions of the increased computational capability would not be tolerated.

In sum, potential economic benefits provide, and will continue to provide, powerful pressures to introduce more digital technology, especially in the form of stored program computers, into future civil aircraft. To many it is almost an article of faith that the introduction of more digital technology into avionics will have a positive effect upon the reliability of the avionics. While the presumption may not be too difficult to demonstrate correctly for the limited cases of device per device substitution, it remains to be determined with *acceptable confidence* that the maximum performance benefits to be associated with the use of stored program com-



puters can be achieved without an intolerable detrimental resultant effect upon the reliability of the total aircraft.

## RELIABILITY REQUIRED OF AVIONICS

The reliability required of a total aircraft system provides a bound on the reliability required of the avionics and the software embedded in it. There are several, not totally unrelated concerns which give rise to aircraft reliability requirements. They are (a) cost of operations for airline operators, (b) disruption of the air transportation system, and (c) safety of occupants of aircraft and individuals on the ground. The concern about disruption of the air transportation system is ordinarily far less prominent than the other two concerns, if it is recognized at all. With its potential for national economic chaos, it could become the source of the very strictest reliability requirements when the nature of design inadequacies and software (un)reliability is considered. However, in this paper as in society in general, safety is recognized as the concern generating the most stringent requirements since clearly the most conspicuously undesirable malfunctions in civil aircraft are those which result in sudden, often spectacular loss of human life.

A minimum acceptable level of safety is specified in regulatory agency directives by the use of the expression "extremely improbable"<sup>2</sup> to describe the likelihood of catastrophic events. However, the requirement is subject to interpretation. A typical interpretation by a major airframe manufacturer is the following:

"... a number less than or equal to  $1 \times 10^{-9}$  has been imposed ... to represent the probability of an event designated as extremely improbable. ... Loss of the CCV/FBW function, given a fault-free system at dispatch, shall be extremely improbable."<sup>3</sup>

There are two points to be noted. First, the qualification "given a fault-free system at dispatch," which is intended to exclude having to consider physically degraded systems in the determination of a system's reliability, begs the issue of software bugs and, indeed, of all questions of design inadequacies. Of course, such flaws are not denied. But they are not included within the reliability computational process. The working hypothesis is that they will be exorcised before operational use of the systems to an extent such that their presence is negligible, or more precisely, such that the frequency of malfunctions of a system due to such residual flaws is sufficiently less than the frequency of malfunctions due to physical degradation to permit them to be ignored in reliability calculations. Conventionally the hypothesis is justified by system verification, i.e., testing, prior to operational use.

The second point to note is that the interpretation applies the requirement for "extremely improbable" events to loss of a specific function critical to the flight of an aircraft rather

than to the loss of an aircraft. No apportionment of the (un)reliability among subsystems is indicated; the occurrence of any malfunction from the set of all malfunctions of the stored program computers whose consequences include loss of the CCV/FBW function must, therefore, be an "extremely improbable" event.

There are also in circulation drafts for an FAA Advisory Circular on the topic of system design analysis to generate a consensus explanation of the expression "extremely improbable." One contains the statements:

"Systems, considered separately and in relation to other systems, should be designed . . . such that a catastrophic failure condition is extremely improbable. . . . Extremely improbable refers to events . . . with a mean frequency in the order of  $1 \times 10^{-9}$  or less per flight or flight hour. Such events are the loss of a number of lives and/or destruction of the aircraft."<sup>4</sup>

The last sentence is explicit; the mean frequency magnitude,  $1 \times 10^{-9}$ , is to be coupled with loss of lives (or aircraft), not with a function. Thus to the extent that malfunctioning computational systems (i.e., stored program computers *including* the embedded software *and* firmware *and* other residual system design inadequacies) can singly cause catastrophic consequences, the occurrence of any malfunction from the set of all such malfunctions must be at least "extremely improbable," and possibly even less likely in order to allow for the apportionment of some of the (un)reliability to other aircraft subsystems—including the human factors.

Finally, statistics on the state of civil aviation safety lend credence to the reasonableness of the interpretations cited above of the regulatory agency safety requirement. Figure 1 contains the history by calendar year of the "average aircraft's average speed" in the recent past. As indicated by the graph, while the period prior to 1974 was a time of transition, the period from 1974 to the present (1978 was the last fully documented year at the time of writing) has been quite stable. The "average aircraft's average speed" has varied from its mean value during the period by no more than approximately 0.3 percent while the total hours flown each year has remained relatively constant, approximately 6.3 ( $\pm 7$  percent) million hours. Therefore, the period from 1974 to 1978 is here adopted as the base period.

Figure 2 contains the history of the mean frequency of fatal accidents (per million hours flown) per calendar year. During the years of the base period, the mean frequency of such catastrophic events has varied between  $0.5 \times 10^{-6}$  to  $1.5 \times 10^{-6}$  per flight hour with a mean mean of approximately  $0.9 \times 10^{-6}$ . Moreover, an examination of individual accident records reveals that the majority of the accidents are not ascribed to equipment malfunctions as the primary cause. Exact proportions are debatable owing to reporting differences; however, it suffices to note that it could be argued that the mean frequency of fatal accidents due primarily to equipment malfunctions was in the range from  $1 \times 10^{-7}$  to  $1 \times 10^{-8}$  per flight hour during the base period. Certainly, other things unchanging, nothing less safe than what is already available is acceptable.

\* CCV/FBW = Control Configured Vehicle/Fly by Wire

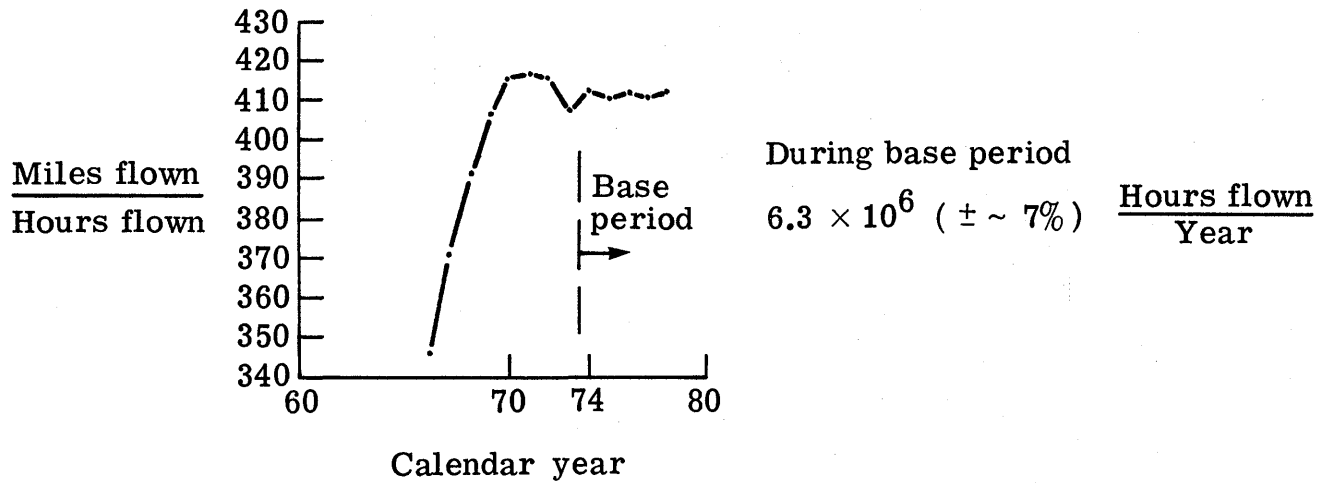


Figure 1.

Hence, for avionics a maximum mean frequency of catastrophic malfunctions of  $1 \times 10^{-9}$  per flight hour over the lifespan of aircraft is asserted to be the reliability requirement. The magnitude  $1 \times 10^{-9}$ , however, is a source of difficulty for it makes a reliability estimate with useful confidence bounds virtually impossible to obtain by conventional system verification prior to operational use because of the number of trials and elapsed time required. In particular, the working hypothesis mentioned above must be justified by some other means—if it is to be relied upon.

A solution often referred to involves the concept of (aircraft) systems of greater reliability constructed from sub-systems of lesser reliability (reference 5 for the theoretical notion)—for example, the use of back-up systems. But, in addition to the still present difficulty of credibly estimating the extremely high reliability of the decision logic implemented to switch to a back-up system, the notion requires the use of alternate systems, external and redundant to avionics. It is precisely such systems which advanced avionics is intended to obviate—if promised benefits are to be

realized. Therefore, the notion is considered inconsistent with the intent of advanced avionics.

#### FAULT TOLERANT AVIONICS COMPUTERS

Computer architectures have been developed specifically in anticipation of the need to satisfy the safety requirements implied by the expression “extremely improbable” discussed above and in anticipation of the data processing needs of future civil aircraft.<sup>3,6,7</sup> The reliability of the physical component devices available now and in the foreseeable future, devices such as processors, memories, power supplies, etc., whose mean time to failure (MTTF) parameters are realistically in the range from  $10^2$  to  $10^5$  hours, implies mean frequencies of *failure* in the range from  $1 \times 10^{-2}$  to  $1 \times 10^{-5}$  per hour for conventional, *fault* intolerant computer systems constructed from such devices. What is somewhat experimental in the referenced architectures is the attempt to attain extremely high system reliability by means of *fault* tolerance

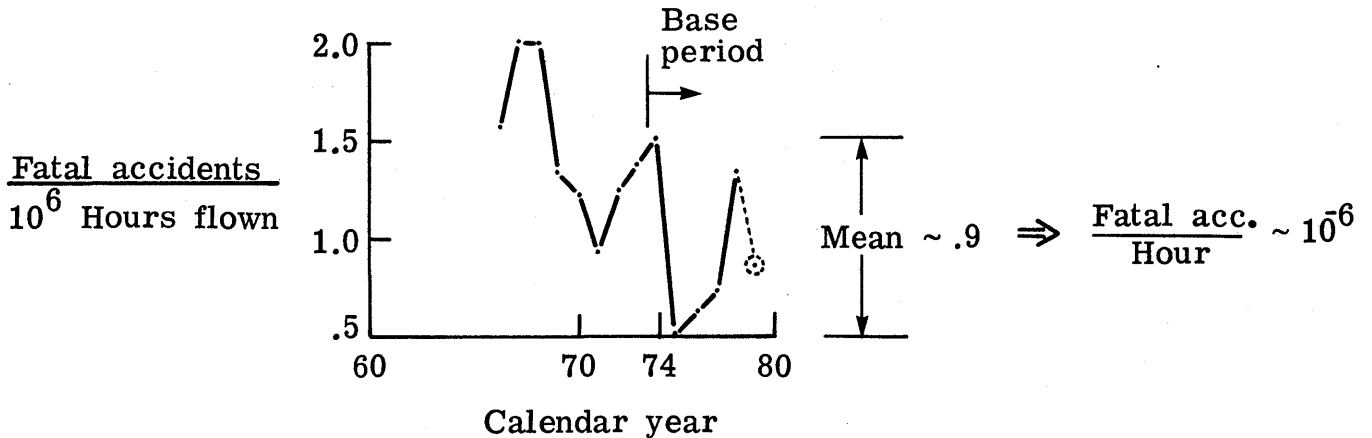


Figure 2.

achieved by the use of redundancy, *error* detection achieved by voting among redundant components, and reconfiguration—all performed internally.

An aside is needed at this point to ensure consistent interpretation of the terms “failure,” “fault,” and “error.” Confusion can arise as a result of the “software” trend, evidenced by recent articles on software reliability,<sup>8</sup> of using the words in a manner reversed from the usage generally adopted for hardware. The following meanings are used here for both hardware and software:

A *failure* is the event when something causes a device, component, system, algorithm, etc., to change its state from one in which it performs its intended function to one in which it does not. The something which causes the change may or may not be known. After the failure, the device, component, etc., is called a *failed* or *faulty* device, component, etc. Any higher level system of devices, etc., which cannot perform its function because a subdevice, sub-etc., is failed is also called failed or faulty.

A *fault* is the particular condition or flaw in a failed device, etc., which differentiates it from its unfailed state.

When the function or output of a device, etc., differs from its intended function or output, that difference is called the *error*. In data processing systems, error means bad or wrong data. An error is all that can be detected internally to a computing system. A higher level system which contains a failed device, etc., emitting errors yet continues to perform its function is said to be fault tolerant. An accumulation of errors may well be the cause of a failure of a higher level system.

Thus a physical device fails when it “breaks down.” Thereafter it contains a fault. A system designer or software programmer can create a design or software containing a fault; in this sense the designer or programmer failed. A fault may or may not be active; when it is, one or more errors result. A fault is latent, transient, intermittent or permanent dependent upon the manner in which it generates errors. A software bug may not surface until some time after a system has been in operation, i.e., it may be *latent*. A bug may cause a data error only occasionally in response to specific, infrequent input data patterns, and may thus appear *intermittent*. Customarily a software bug is regarded as a *permanent* fault, remaining in the system ever after from the moment of its creation by a programmer. However, it is possible for a bug,

having given rise to a data error, to disappear from an operational system—in which case it appears as a *transient*; the resulting bad data may or may not be attenuated in further processing. As an example, consider the common occurrence of failing to preset a variable at system start up. Thus, software bugs can appear to share the possible attributes of hardware faults.

The referenced computing systems are designed to detect and contain errors and isolate faults in physical components at the level of processors, memories, etc.—generally. Detection requires at least comparison; containment and isolation require a plurality. Necessary algorithms are implemented in hardware *and* software. When components are deduced to be faulty, they are ignored in future computations by the unfailed components—not unlike ostracism within a social system. Functions previously performed by components since failed are distributed among the unfailed components; if there is insufficient computing capacity remaining, those functions least important to the flight of an aircraft are discarded. The process continues until insufficient resources remain to perform minimum computations and the system cannot support flight control.

The state transition diagram in Figure 3 is a simplified representation of the scenario above, incorporating the essential approximating assumptions made in reliability analyses to date of highly reliable fault tolerant avionics computing systems. The analyses have been more complete and searching than this simple representation—accounting for various component types, not all interchangeable, having different propensities for failure, etc. Yet the additional refinements of analysis do not significantly modify the conclusion below.

The prime assumption in the reliability analyses is that the elemental failures at the physical level in any given component occur independently of the occurrence of other such failures in other components. It is assumed, and every effort is made to ensure, that the environment is controlled such that “massive,” system-wide failures do not occur. For example, avionics systems must be protected from lightning discharges having such system-wide effects. The diagram in Figure 3 represents this assumption by restricting degradation solely to a state with exactly one fewer component.

A second assumption is that the frequency of nearly si-

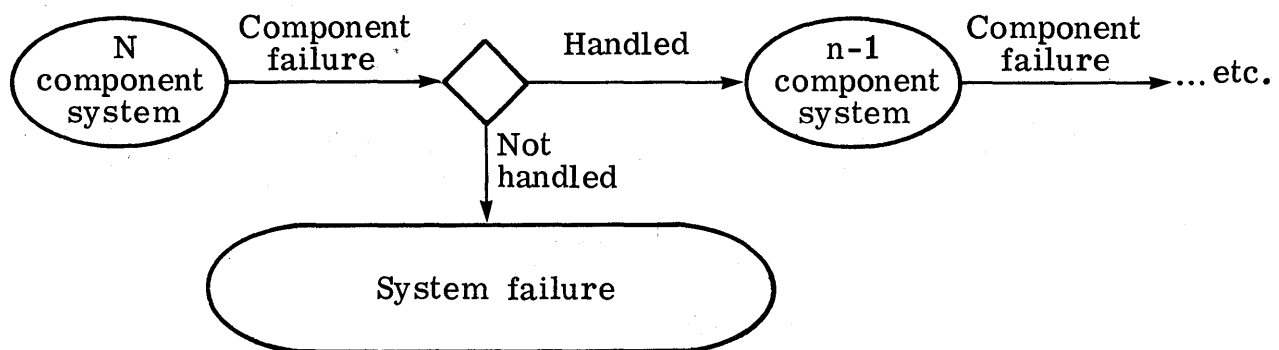


Figure 3.

multaneous errors (resulting from different failures) is sufficiently small to be neglected in the count of system failures. To the extent to which this assumption is not correct, the algorithms which perform detection and containment of errors and isolation of faults are inadequately designed for they cannot cope with many combinations of multiple failures concurrently. Clearly, the greater the latency time between the creation of a fault and its manifestation as an error, the less justified the assumption. Yet the assumption is maintained for its mathematical convenience and for lack of sufficient hard data (to date) to support alternate models of behavior. The instances when the assumption is not correct are accounted for and represented in the diagram by "coverage" parameters, conditional probabilities that, given a failure, transition to a correctly reconfigured and operating state is successfully accomplished.

A third assumption is that, once reconfiguration has been performed, any further errors generated by the faulty component are prevented from propagating outside predetermined containment boundaries and thus prevented from causing secondary failures. This assumption is also represented in the diagram by means of the restriction of degradation solely to a state with exactly one fewer component; in addition, analyses normally constrain component failure rates to be independent of system state.

Finally, analyses of the avionic computer systems have conventionally neglected software and design faults—hypothesizing a system fault-free, the bugs exorcised by much testing and program correctness proving and perhaps even entirely avoided by application of disciplined management and program development techniques. It should be noted that, because of this "decoupling" of the software (un)reliability from the process of estimation of computer system reliability, the notion of a required software reliability becomes disassociated from the context of the application. Denied this direct, measurable relation to an application, rather than remaining simply a characterization of software's merit, the notion is often associated with comparisons and orderings of methods for *implementing* software (e.g., preferences for certain program structures, for estimating number of bugs remaining in code, etc.).

#### SOFTWARE AND DESIGN LOGIC AS SYSTEM ELEMENTS

While the number of faults (flaws) remaining after careful development and testing remains problematical, what is important in the context of an application and reliability are the frequency with which faults are activated and the severity of the consequences of the errors generated (emphasizing again the context of the application). If software programs (and design logic) are considered as system elements, possible sites of residual faults, interacting with other more tangible components and capable of leading to avionics computer system failure, then the real consequence of software malfunctions can be evaluated and the reliability required of software can be stated.

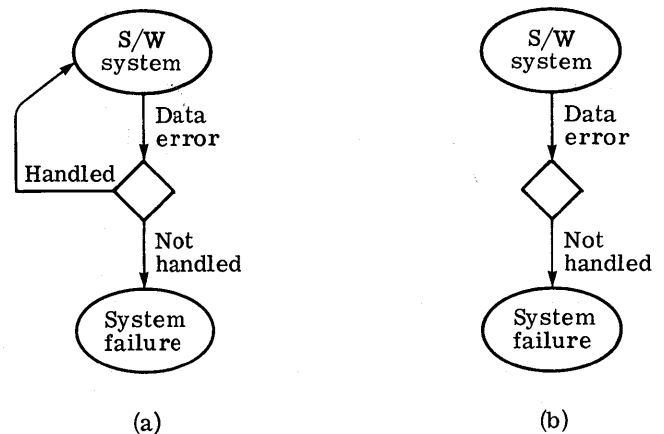


Figure 4.

A fatuously simple representation of software behavior is illustrated in Figure 4a; it illustrates a difference between hardware and software. Unlike the case for hardware in which redundancy is provided by replications of components, simply replicating software in replicated components only replicates any faults; consequently, errors occur in replicate sets and the notion of error detection and fault tolerance by comparison and majority voting is defeated. (The same is true of design logic.) Yet software fault tolerance techniques, of which B. Randell and his colleagues at the University of Newcastle-upon-Tyne have been leading innovators, exist. They attempt to provide "redundancy" by means of alternate, secondary algorithms and "acceptance" tests to detect errors. Such concepts appear applicable to avionics; minimal additional time and memory usage are required.<sup>9</sup> Accordingly the state transition diagram in Figure 4a represents the behavior of fault tolerant software on the assumption that successful recovery from a software error is followed by return to an initial (software) state. That is, unlike hardware, software may not degrade. The rationale for the assumption is that a fault responsible for a software error has always lain latent; presumably it will do so again after the date or conditions which activated it have passed. A recovery parameter, analogous to the "coverage" parameters mentioned for hardware above, can be used to account for the possibility of not detecting or recovering from all software errors. Figure 4b is an equivalent but simpler representation.

Studies of system failures due to software have been published, e.g., some recent data indicating that for one special application and for one failure mode a hypothesis of exponentially distributed system failure times due to software was *not* tenable,<sup>10</sup> but there is no credible, empirical evidence for the selection and justification of any complex, general model of system failures due to software<sup>11</sup> let alone due to general design flaws. Figure 4 is not intended to suggest that a simple model is sufficient for analysis and prediction purposes.

The diagram in Figure 5 is a combination of Figures 3 and 4b to represent a total system comprising hardware and soft-

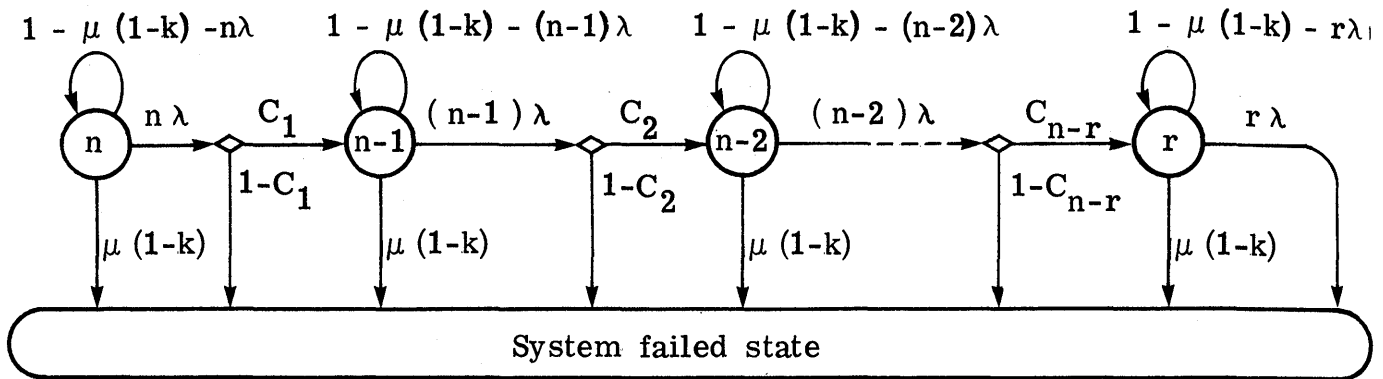


Figure 5.

ware (and design logic). An initial  $n$  redundant hardware component system can fail by either

- (1) suffering a hardware component failure and degrading (with conditional probability  $c_1$ ) to an  $n-1$  hardware component system which can in turn fail, or
- (2) suffer a component failure from which it cannot recover (i.e., with conditional probability  $1-c_1$ ), or
- (3) suffer a software error from which it cannot recover.  $\mu$  represents the rate of fatal software errors;  $1-k$ , the conditional probability of not being able to handle an error.

Clearly, each and every path to failure is bounded by the  $1 \times 10^{-9}$  reliability requirement. In particular, for  $\lambda$  and  $\mu$  interpreted as mean frequencies of hardware and software critical malfunctions per hour,  $n\lambda(1-c_1)$  and  $\mu(1-k)$  must each be less than approximately  $1 \times 10^{-9}$ . For reasonable and realistic values of  $n$  (3 to 10) and  $\lambda$  ( $\sim 10^{-4}$ ),  $.999997 \leq c_1 \leq 1$ , in a sense, a requirement on design logic. No credibly reliable estimates for  $\mu$  are available, hence it can only be required of software that  $\mu(1-k) \leq 1 \times 10^{-9}$ .

Since, and if,  $\mu$  and  $(1-k)$  both pertain to the same kind of failure, i.e., software bugs and design flaws, one might speculate that they are similar in magnitude and that a credible demonstration that  $\mu$  and  $(1-k)$  are each approximately  $10^{-4}$  or  $10^{-5}$  is the reliability requirement for avionics software.

## CONCLUSION

To the extent to which the assumptions stated above approximate the real world, hardware can be replicated until required reliability is achieved, but the same is not true of software. Hence, software is the critical element of highly reliable (fault tolerant avionics) computer systems; the prob-

lem of design inadequacies is considered to be the same as the software problem. Since data on software error rates (in the precision implied necessary by the model above) are lacking, it is currently not possible to predict with "credible" confidence that a highly reliable software system will indeed satisfy its reliability requirement. This leaves an avionics system with embedded software in an uneasy state but points quite clearly to the area of needed research.

## REFERENCES

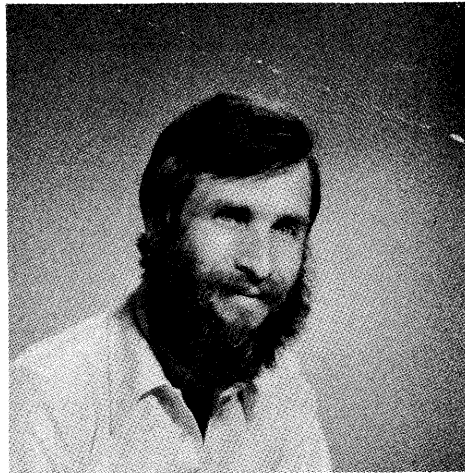
1. *AVIONICS: Projections for Civil Aviation, 1995-2000*, NASA-ASEE 1979 Engineering System Design Fellows, Old Dominion University, NASA CR-159035, September 1979.
2. Federal Aviation Administration FAR part 25, paragraph 25.1309(b), dated 5 August 1970.
3. Bjurman, B. E., Jenkins, G. M., Masreliez, C. J., McClellan, K. L., and Templeman, J. E., *Airborne Advanced Reconfigurable Computer System*, Boeing Commercial Airplane Company, NASA CR-145024, August 1976.
4. Federal Aviation Administration Advisory Circular No. 20-draft, "System Design Analysis," undated, initiated by AFS-130.
5. Barlow, Richard E. and Frank Proschan, *Statistical Theory of Reliability and Testing*, Holt, Rinehart and Winston, Inc., 1965.
6. Hopkins, A. L., Smith, T. B., and Lala, J. H., "FTMP - A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft," in *Proceedings of the IEEE*, Vol. 66, No. 10, pp. 1221-1239.
7. Wensley, J. H., Lampion, L., Goldberg, J., Green, M. W., Levitt, K. N., Melliar-Smith, P. M., Shostak, R. E., and Weinstock, C. B., "SIFT: The Design and Analysis of a Fault-Tolerant Computer for Aircraft Control," in *Proceedings of the IEEE*, Vol. 66, No. 10, pp. 1240-1254.
8. "Special Issue on Software Reliability," *IEEE Transactions on Reliability*, Vol. R-28, No. 3.
9. *Fault Tolerant Software for Aircraft Control Systems*, The Aerospace Corporation, NASA CR-145298, February 1978.
10. Beaudry, M. D., *A Statistical Study of Service Interruptions at the SLAC Triplex Multiprocessor*, Technical Report #141, Computer Systems Laboratory, SEL 79-006, Stanford University, May 1978.
11. Thibodeau, R., *The State-of-the-Art in Software Error Data Collection and Analysis*, General Research Corporation, Army Institute for Research in Management Information and Computer Science, Georgia Institute of Technology, AIRMICS Contract # DAAG29-76-c-0100/0598, 1978.

## Languages

This year the "Languages" technical area focuses on practical matters. The four sessions are all concerned with language issues arising from real world considerations.

"Ada, Where it Stands Now" will assess the current status of the new Department of Defense programming language for embedded computer systems. Embedded computer systems are those which interface with non-computer devices such as satellites and submarines. "MUMPS" is considered by its devotees to be an instant data base implementation system. This session will spread the word. "High Level Languages for Microprocessors" explores the availability, outlook, and problems of powerful languages for little computers. "Pascal in the Real World" demonstrates that a well designed language can be used to solve real world problems.

If a common theme emerges from these sessions, it is that it is possible to design, implement and use clean, powerful and "academically" acceptable languages for the nitty gritty jobs which must be done.



Russell J. Abbott  
*Area Director*



# A linguistic comparison of MUMPS and COBOL

by THOMAS MUNNECKE

Veterans Administration Hospital  
Loma Linda, California

"I speak Spanish to God, Italian to women, French to men, and German to my horse." *Charles V of France*

## COMPUTERS AND LANGUAGES

There are endless discussions in data processing circles about which computer language is best. Not surprisingly, the arguments generally boil down to each participant saying: "The language I know is best." These dogmatic beliefs often lead to vigorous debates among programmers who use different languages.

Languages have a deep relationship to the thought patterns of their users. If a programmer can't easily say something in a computer language, he is not inclined to think it, either. When programmers of the different languages meet, they are projecting their thought patterns into each others' language. Finding that the language does not express these thought patterns as richly as their native language, they often judge the other language as inferior.

Languages and their user communities tend to grow together. A user community cannot be expected to change its language unless it sees a new set of needs not met by its current language. Contrary to Charles V's fluency, it would be difficult to convince a German that he should learn Italian to speak to women. However, he could be convinced to learn the language of mathematics when he realizes that his spoken language is not sufficient for his mathematical needs.

MUMPS was created as a computer language in response to a new group of needs. MUMPS was designed to be a simple, small-computer-oriented language dedicated to a specific task. It turned against the trend toward disintegrative languages which grew out of early batch processing techniques. It pursued a new dimension of user/computer interaction. A dimension seldom seen, much less appreciated, by typical COBOL programmers.

Today, there is a crushing new group of needs which modern information systems must face. People costs to run computer systems far exceed computer costs. In order to recognize this linguistically, perhaps we should rename "computer systems" to "people systems." "Computer languages" should become "people languages." These terms more accurately reflect the needs of modern information

systems. The computer is merely a medium by which an organization achieves its goals.

Viewing the computer as a medium is an effective way of reflecting these changing needs. The computer would then be in the same general category as books, magazines, television or telephone.

People might then view computer programmers turned computer scientists in the same light as a television cameraman who calls himself a television scientist or a printer who calls himself a book scientist.

This new group of needs can only be solved by the people who brought them about—the users of the information system. They cannot expect "computer scientists" to solve their problems any more than they can expect a printer to write a book for them.

The role of the computer must be to linguistically support the user in his own terms, adapting to his needs, and being as forgiving and friendly as possible. This, of course, is a flagrant violation of oldtime wisdom, where one could not waste computer time on such frills.

MUMPS takes this role of user adaptability seriously. The natural logic of the language encourages programmers to turn control over to users with simple, yet powerful commands. MUMPS users tend to show an almost reverent attitude to their systems.

Conversely, COBOL was designed in an era when computers were expensive, programmers were cheap, and only science fiction buffs dreamed of small computers. The natural logic of COBOL is oriented towards batch processing of fixed records. As people struggled to make COBOL adapt to the needs of on-line systems, they added features\* such as data base management systems, data communications monitors, message formatting monitors, and the like. Rather than adding flexibility to COBOL, all of these disintegrative appendages have stifled the language.

MUMPS, perhaps, can best be understood by the appendages which it lacks. This lack of features is MUMPS' great-

\* The term "feature" has sometimes been defined as a "design flaw which marketing" has noticed. For example, a computer manufacturer once announced a distributed processor which had the "security feature" that it could not be programmed locally. Only cross-compilations downloaded from the host processor were allowed. Cynics who felt that this feature was really a cover up for not having local software were vindicated some time later when the manufacturer announced a new feature—local programming ability.



est strength, not a weakness, as the old schools would have it. All of the following functions of these appendages are integrated into MUMPS' unique symbolic structure:

- Assembler Language
- Compiler
- Data Base Management System
- Sort/Merge Utilities
- Job Control Language
- Linkage Editor
- Debugger
- Data Communications Monitor
- Core Image Dumps
- Absolute Addresses
- Type Definitions
- Dimension Statements
- Message Format Processor.

MUMPS is an entire data management system, built within a single linguistic framework. All of the utilities, control blocks, and organizational paraphernalia which are considered "features" of COBOL-based systems are simply not needed with MUMPS.

In order to facilitate comparison, IBM's data base management system, IMS 360 (Information Management System) has been selected as a typical add-on to COBOL for data management applications.

This elegant simplicity of MUMPS is generally not appreciated by newcomers examining the MUMPS syntax. For example, the up-arrow is the only indicator in the MUMPS language that the program is working with a data-base record instead of a local array variable. A reader who was expecting a long list of data-base manager subroutine calls will be disappointed. He should not, however, condemn the language for successfully eliminating those appendages of archaic languages.

## INTRODUCTION TO MUMPS<sup>1</sup>

MUMPS, the Massachusetts General Hospital Multi-Programming System, is a high-level interpretive programming language and data-management system. It is particularly suited for interactive applications which require a large, shared dynamic data-base and the efficient manipulation of textual data.

The development of MUMPS began at the Laboratory of Computer Science, Massachusetts General Hospital, Boston, Massachusetts, in 1966. Previous research in medical information systems had encountered frustration and dissatisfaction with the current state of the technology. The experiments during 1960 to 1965 with assembly language systems were usually unsuccessful due to long development time and excessive turnaround times for even the most simple program modifications. For these reasons, the staff of the Laboratory of Computer Science set out to design an efficient time-sharing system for clinical data management.<sup>2,3</sup> The characteristics of this system were patterned after JOSS, a high-level, interpretive language developed at the

RAND Corporation in 1964. Experience was also drawn from descendants of JOSS such as TELCOMP and STRINGCOMP developed by Bolt, Beranek and Newman, and FILECOMP specified by General Electric for the MEDINET system.

The goal of the MUMPS system was to combine a simple yet powerful high-level language with an easy-to-use data-base handling system. The MUMPS Language was designed to be easy to learn, with simple methods of program creation, modification and debugging. Language capabilities for the handling of variable length character strings and multi-terminal I/O were also required. A sparse hierarchical data-base system was developed as an integral part of MUMPS. The hierarchical structure was determined to be the most appropriate method of handling the complex of demographic data, diagnosis, laboratory results and other data required for clinical data management. A data-base handler was designed to facilitate access to the hierarchical structure from the MUMPS language. Basic features were developed to implement symbolic update and retrieval functions. The design emphasized the support of a dynamic data-base, subject to frequent updates interspersed with on-line queries. The MUMPS programmer could freely design both the content and structure of data to best fit his application. Finally, the system was implemented with a compact, time-sharing executive to make efficient use of all resources in a mini-computer environment.

The appropriateness of these design goals is now well documented. The MUMPS system has found a significant place in both medical and non-medical environments. Since the original implementation at the Laboratory of Computer Science, at least seven other dialects and variations of MUMPS have been developed. In order to rejoin these divergent views and promote program interchange, the U.S. Department of Health, Education and Welfare, and the National Bureau of Standards initiated development of Standard MUMPS. This standard specification, along with other MUMPS documents for language teaching and translation was accepted by the American National Standards Institute in 1977.<sup>4</sup>

MUMPS is more than a programming language. It is a linguistically integrated data management system combining with a single syntax what other operating systems might call: (a) an application programming language; (b) a job control language; (c) a linkage editor; (d) a data-base management system; and (e) a data communications monitor.

### *The MUMPS storage hierarchy*

MUMPS goes beyond traditional data management by allowing records (nodes) to be interconnected hierarchically. Thus, a node can be the child of another node and the parent of any number of nodes. Networks are not allowed: each node can have only one immediate parent (though there can be any number of generations in the structure), and no node can have a parent as a descendant. It is also possible to have nodes which serve only as connectors, i.e., which contain no data. These are often called *pointers* though they should

not be confused with the traditional use of this term. The set of subscripts or keys which define a node is actually the path from the top of the hierarchy to the node. Thus, there is a single key associated with the descent from a parent to a child. MUMPS provides capabilities for addressing a node, for the purpose of storage or retrieval by the complete sequence of keys, or by a portion of the sequence. When a portion is used, the missing keys are supplied by the system based on the most recent reference to the database. It is also possible to test whether or not an arbitrary node exists and whether or not it has any data. Also, there is a means of sequencing among a set of siblings, i.e., given an arbitrary position within the hierarchy of finding the next numerically higher single key. This technique cannot be used to cross from one set of siblings to another. In general, there are capabilities for moving from parent to child and from sibling to sibling, but the only way to move back up is to begin again from the top.

This scheme of node interconnection is very useful because many real world systems can be represented by a hierarchy or something simpler (e.g., a hierarchy includes a simple indexed organization as a subset). Thus, a complex entity may be represented as being constructed of subentities, each of which is in turn sub-divided. In addition, there is frequently a need for network structures. These relationships can be created by the applications programmer. For example, although MUMPS does not itself provide inverted files, MUMPS programs construct such files by storing logical references to other nodes. For example, in an inventory control system a parts file might have part-number as a key. A file of suppliers could logically point into the parts number file by means of the part numbers.

People often ask of MUMPS systems: "Why does it not support other languages?" "Why does it not compile its code?" These are a sampling of hundreds of similar questions as to why MUMPS does not support the features of the language and operating systems to which they are accustomed. These questions are of the variety: "When did you stop beating your wife?"

MUMPS' only retort to these questions is: Why are all these features needed? Are the 12 access methods used by a typical COBOL/IMS system a strength or a weakness? Is the linguistic distintegration characteristic of almost all "modern" operating systems solvable by adding more fragmentary features? Is the increasingly complex combine of pre-compiled object modules, control blocks, and linkage editors really suited to today's online environment?

### Standards

A standard is only as strong as its weakest linguistic link. A standard which ignores major linguistic structures such as terminal communication, data-bases, and the like necessarily will force users into an integration crunch and a reversion back to disintegrative designs.

The American National Standards Institute MUMPS standard is a linguistically strong standard. Programs and data-bases written in standard MUMPS are assured of port-

ability because MUMPS requires no linguistic support from all of the peripheral languages common to older operating systems.

Oddly enough, this linguistic independence is often the most vocally criticized aspect of MUMPS. People reviewing MUMPS, after noting its lack of features, go on to criticize its "incompatibility" with current systems. One reviewer's "incompatibility" is another's "linguistic purity." The fact that MUMPS' designers refused to disintegrate their linguistic realms to archaic operating system features should not be held against them.

### Access methods

The list below compares the access methods used by MUMPS and COBOL using IBM's Information Management System (IMS).

COBOL/IMS	MUMPS
QSAM	GLOBALS
BSAM	
QISAM	
BISAM	
BPAM	
VSAM	
BTAM	
HSAM	
HISAM	
HIDAM	
HDAM	
OSAM	

While this incomplete list of 12 access methods may look impressive, it raises the question: Are all these really necessary? To the MUMPS programmer, of course, the answer is no. Even the term "access method" is foreign to him.

### Pointers

Today's disk storage technology requires internal pointers for efficient data management. The following list shows the difference between COBOL/IMS and MUMPS.

COBOL/IMS	MUMPS
Physical Parent	\$NEXT
Physical Child	\$DATA
Hierarchical Forward	
Hierarchical Backward	
Physical Twin Backward	
Logical Twin Forward	
Logical Twin Backward	
Physical Child Last	

The differences between these two columns run far deeper than is apparent. While the MUMPS program is able to dynamically structure its search, based on the content of the

data it finds as it works its way through the data-base, the COBOL/IMS pointer system is rigidly defined in pre-compiled control blocks which may be changed with only the greatest caution.

MUMPS' richness in content-oriented data structuring is possibly best illustrated by the fact that MUMPS can represent a data value of "nothing." Whereas COBOL would require a bogus value, for example 999 or spaces, MUMPS simply recognizes a null data value. This is roughly equivalent to the discovery of zero in algebra.

### Languages

People often say "We are a COBOL shop." However, due to the inherently weak nature of COBOL and the languages which have sprouted up around it, many languages are used within the COBOL environment.

#### COBOL/IMS LANGUAGES

Language	Pages of Documentation
COBOL	300
Data Language/1	100
Job Control Language	200
Linkage Editor	100
Message Format Services	100
System Definition MACROS	300
Assembler Language	200
MACRO Assembler	100
Program Specification Blocks	150
Data-Base Definition Blocks	150
Total Pages	1700

Each of the languages listed above has its own linguistic domain, reference language, and documentation. This has a profound effect on the overall operation of the computer support staff. Specialists have arisen, spreading the responsibilities of a given system over a multitude of people, such as systems programmers, data-base administrators, network administrators, and control block librarians, in addition to the traditional programmers and systems analysts. Listings of code in each of these languages is thus spread out over each of these specialists, and typically jealously guarded. Thus, a programmer searching for the cause of an error may spend a good portion of his time searching for specialists and/or their listings.

### Error detection and correction

Try as we may, errors will occur. MUMPS, being an interactive system, will display the error with an explanation directly in the MUMPS language. The programmer may examine variables, modify the program, and resume processing directly from the keyboard, all using MUMPS language. Thus, all communication is accomplished in one language, at a terminal, as soon as the error occurs. This is not the

case with the COBOL/IMS environment. An error begins with a hexadecimal dump (typically, 25-50 pages long), which the programmer sees hours or days after the error occurs. He must thread these numbers through a maze of languages, specialists, and listings. Following is a highly polished translation of one error which might occur in a COBOL/IMS environment:

### COBOL/IMS error detection:

"An Assembler Language *Dump* shows that the *COBOL Return Code* of the *PCB* of the *PSB* defined in the *PARM* of the *JCL EXEC* card (or specified in the *IMS SYSTEM*) generated by the *PSBGEN* utility contains a *PCB MACRO* with *SEGEN MACRO* parameter *PROCOPT* incompatible with *DL/I* call *FUNCTION* parameter."

This particular error traversed six languages, on four different listings, requiring working knowledge of 1000 pages of documentation. Furthermore, the error was presented to the programmer after the fact, perhaps after irreversible conditions had changed things.

With the functions of the data-base management system, job control language, control blocks, utilities, assemblers, compilers, and communications monitors stripped away, one arrives at a comparison of the entire COBOL language and a portion of MUMPS.

### CASE COMPARISON-PAYROLL PROGRAM

The author once translated a COBOL program into MUMPS. The COBOL program was part of a payroll system which received a batch of time and attendance records and computed the gross and net pays, various leave balances, and the like. The MUMPS version replaced the batch system with on-line data entry and validation, with immediate computations. Thus, the MUMPS version had more work to do.

ITEM	COBOL	MUMPS	PERCENT
Lines of Code	3600	300	8
"IF" Statements	460	89	19
"GOTO" Statements	650	43	6
Total Program Size	120K	9K	8

The MUMPS version required approximately 8% of the number of lines of code, 19% of the number of conditional checks (even with the added validity checks), 6% of the program branches, and 8% of the run-time memory.

Execution time on a \$100,000 MUMPS minicomputer was approximately twice as long as the several million dollar COBOL/IBM 370/158 computer. Exact programming times were hard to estimate, but three weeks were spent on the MUMPS version, while the original COBOL version took an estimated six to nine months.

### CASE COMPARISON—MESSAGE DISPLAY

To illustrate these differences, a portion of a COBOL program was selected which writes out the message: "Affidavit

XXX processed, Precinct is YYY." The MUMPS version of this simple message display program is: WRITE !, "AF-FIDAVIT ", AFFNO," PROCESSED. PRECINCT IS ", PREC

The COBOL version illustrated below requires the message to be formatted in the DATA division. (Note that the 'W' is actually a special character, which must be multi-punched on a keypunch. Thus, the keypunch, printer, and computer all have different understandings of the same character):

Data division:

```
03 FILLER PICTURE X(10) VALUE 'AFFIDAVIT'.
03 MSG-AFF-NO PICTURE X(7) VALUE SPACES.
03 FILLER PICTURE X(10) VALUE 'PROCESSED'.
03 FILLER PICTURE X(13) VALUE 'PRECINCT
IS'.
03 MSG-PREC-NO PICTURE X(5) VALUE ''.
03 FOB PICTURE X VALUE 'W'.
```

Then the message must be transmitted in the procedure division.

Procedure division:

```
000-ENTRY.
MOVE H-PREC-NO TO MSG-PREC-NO.
MOVE H-AFF-NO TO MSG-AFF-NO.
MOVE CMPL-MSG TO OUT-SEG-1.
TERM-TRAN.
CALL 'TELECALL' USING DECB-ADDR,
TPTRNSMT, TP-SW.
RETURN-TO-VRNEWAFF.
```

The MUMPS version used 6% of the lines of code, and 9.6% of the number of characters. The COBOL version made six explicit declarations of the lengths of fields involved. MUMPS made none.

Comparisons:

ITEM	COBOL	MUMPS
Lines of Code	16	1
Characters	500	48
Number of Bindings	6	0

## DATA INDEPENDENCE

Someone from the disintegrative school might defend it at this point by saying: "But what about data independence?\*" The traditional file structures provide for data-independent programs, through well-defined linkages."

If one examines the situation carefully, one sees that data independence is a mythical construct of the disintegrative school—data dependence is merely being transferred to yet another language or languages. This process is somewhat akin to a doctor "curing" a patient by erasing his symptoms from the medical record.

\* Data independence<sup>3</sup> is defined to be the "immunity of applications to change in storage and access strategy."

Mr. C. J. Date<sup>5</sup> discussed the data independence of IMS. If one examines the darker side of IMS's data independence (i.e., its data dependence on control blocks), things are slightly different. For example, in order to add a single byte to a key field in a COBOL/IMS system, the following procedures must be followed:

- 1) Change the data-base definition block
- 2) Change the program specification block
- 3) Regenerate the accumulated control block
- 4) Change any message output format, message input format, device input format, or device output format block referencing the field.
- 5) Change the data division of each COBOL program which references the field. Furthermore, the procedure sections of each program must be scanned for move statements which overtly or covertly reference the field. Once associated fields have been identified, they too must be scanned for reference to yet other fields. COBOL can covertly reference fields through redefining, corresponding moves, assembler language subroutines, or parameter passing.
- 6) Unload the data-bases under the old data definition.
- 7) Reload the data-bases under the new data definition with a program which shifts the data to its new format. Depending on the size of the data-base, the unload/reload process can take from a few minutes to several days. The data-bases are not accessible to terminals during a major portion of this time.
- 8) Since the changes are so pervasive, prudence dictates that the control blocks, application programs, and data-bases be tested on a duplicate "test" system. Thus, all of the above steps must be carefully sequenced through twice.

Thus, the COBOL/IMS concept of data independence can trigger off a complicated sequence of control block changes, recompilations, job control language changes, and significant data-base down time for the simple process of adding a single byte to a field. The operations staff must use several macro languages, COBOL, job control languages, linkage editor, and manual procedures to accomplish this task.

Many attempts to correct this complicated sequence have been made, including adding *another* linguistic entity such as a master data dictionary. However, this can only serve to further disintegrate linguistic control. As a new language is added it imposes its own (weak) data definition structure, reference language, source language control, etc.

One is tempted to ask, "Why is adding a single byte to a field such a major undertaking for such a sophisticated computer system? Why is every linguistic domain so dependent on exact field length specifications?" The answer is that the disintegrative approaches are built up from compiled logic which uses absolute addresses (or absolute offsets). The languages lose control of the data at the moment of compilation.

MUMPS, on the other hand, makes no such linguistic distinctions. Fields are treated dynamically according to whatever data are found in them. Data-base structures grow and

shrink according to whatever data is stored in them. Reorganization is seldom necessary due to internal techniques of balanced multiway trees.<sup>6</sup> All data references are symbolic; if a field does not exist in a particular instance, it takes no space. There are no data definitions, procedure-scanning, absolute addresses, REDEFINES, assembler language sub-routines, control blocks, or data set definitions to worry about—they simply do not exist. The only changes a MUMPS programmer may need to make to add a byte to a field are:

- 1.) If an explicit length reference is made to the field, it will have to be changed to the new length. For example, IF \$LENGTH (INPUT) > 6 WRITE "TOO LONG" would have to have the "6" changed to a "7".
- 2.) If the field is printed on a pre-printed form, the output routine may have to be changed.

#### MISCELLANEOUS OBSERVATIONS OF COBOL AND MUMPS

1. To a MUMPS programmer, COBOL appears to be a linguistic flatland in which only the simplest data structures may be expressed. Problems which he dismisses with a simple statement in MUMPS would be pages of code in COBOL.
2. COBOL's rigid structure is its most prominent characteristic. MUMPS is known for its flexible data and program structures. For example, if a COBOL program encounters a 5 digit number to be printed in a 4 digit field, it will change the data to meet the format. MUMPS would rather print the right data in the wrong format than print the wrong data in the right format. In all of the structure/content design tradeoffs, MUMPS stresses content, while COBOL stresses structure.
3. COBOL makes a very strict distinction between "program" and "data." These distinctions are not necessarily made in MUMPS. A MUMPS program could execute a data-base, or a program could be treated as data. This allows MUMPS to be used as an implementation language for higher level languages or systems. It also allows for all of the MUMPS operating system utilities to be written in MUMPS itself, rather than resorting to assembler languages, linkage editors and the like.
4. COBOL is usually compiled, whereas MUMPS is usually interpreted. The COBOL language disappears at execution time. It assumes that the programmer has accounted for all eventualities *before* the program was compiled. MUMPS, on the other hand, is free to make use of the interpreter during the execution of the program.
5. MUMPS takes the "small is beautiful" approach to computing. Originally designed for minicomputers, it exploits the dedicated nature of small computers. It makes heavy use of the cheapest resource (central processor time), and minimizes the most expensive resource (people time). MUMPS systems generally grow

by adding more systems, rather than larger ones. COBOL, on the other hand, grew up in the "bigger is better" school. Manufacturers stressed the "economies of scale" of large computers, saying that a larger computer would work more cheaply per unit of work. These economies have clearly turned around with today's microelectronic technology.

COBOL users and large scale computer manufacturers, fearing loss of control, have often responded by scaling problems up to the point where they can be solved only by large-scale computing equipment. MUMPS users, on the other hand, tend to scale problems down to smaller and smaller computers.

6. The author has a theory that the response time of an interactive computer system increases exponentially with the cost of a computer. This is due to the fact that a computer must be idling along at 30-50% capacity in order to handle unexpected interactive loads. Thus, the cost of good response time is proportional to the cost of "wasting" computer time in reserve for the unpredictable needs of an online system. The owner of a \$500 Radio Shack computer does not hesitate to "waste" computer time to serve his needs, but his techniques would bring shudders to the manager of a large scale computer.
7. There is a controversy in the data processing field titled "Superprogrammers versus Mongolian Hordes." MUMPS supports the "superprogrammer" philosophy. Individuals, or small teams of MUMPS programmers, are capable of producing what large teams of COBOL programmers can do. Few "superprogrammers" are content to remain COBOL programmers. Their talents are frustrated by COBOL's awkwardness, inflexibility, and slow development cycles. Good COBOL programmers tend to be promoted to higher paying positions in the COBOL organizational hierarchy, a clear case of the Peter Principle. In contrast, MUMPS programmers can draw higher salaries due to their higher productivity, and happily remain MUMPS programmers.
8. COBOL programmers tend to exhibit great concern about computer efficiency with a corresponding lack of concern about the efficiency of the users of the system. I have labelled this characteristic "cyclephobia"—an irrational fear of wasting computer cycles. Cyclephobes tend to see problems in light of the primitive operations expressible in COBOL. MUMPS programmers, on the other hand, have a much healthier attitude toward computer/user efficiency tradeoffs. This is partly because they use an inherently more friendly computer—the small computer, and partly because MUMPS naturally directs the programmer to "friendly," responsive computer interactions.

#### REFERENCES

1. Munnecke, T. H., R. F. Walters, J. Bowie, C. B. Lazarus and D. A. Blidger, "Mumps: Characteristics and Comparison with Other Programming Systems," *Medical Informatics*, Vol. 2, No. 3, pp. 173-196, 1977.

- 
2. Greenes, R. A., A. N. Papalardo, C. W. Marble and G. O. Barnett, "Design and Implementation of a Clinical Data Management System," *Computers in Biomedical Research* 2, pp. 469-485, 1969.
  3. Bowie, J. and G. O. Barnett, "MUMPS—An Economical and Efficient Time-Sharing System for Information Management," *Computer Programs in Biomedicine*, 6, pp. 11-22, 1976.
  4. American National Standards Institute, Inc., *American National Standard MUMPS Language Standard*, ANSI 11.1-1977, 1977.
  5. Date, C. J., *An Introduction to Database Systems*, Addison Wesley, Reading, Massachusetts, 1975.
  6. Knuth, P. E., *The Art of Computer Programming, Vol. 3. Sorting and Searching*, Addison-Wesley, Reading, Massachusetts 1973.



# The design of PLAIN—Support for systematic programming

by ANTHONY I. WASSERMAN\*

Section on Medical Information Science  
University of California, San Francisco  
San Francisco, California

## DISCIPLINE IN SOFTWARE DEVELOPMENT

The successful construction of medium and large software systems requires the management of the complexity inherent in the problem being programmed. A well-disciplined approach to software development involves the production of a complete specification, a complete problem solution, and program design prior to the inception of actual coding. In practice, this requires the production of some form of program design representation [1] from the original specification, with the action of each module specified with a program design language [2]. Furthermore, data structures are specified and refined, in some cases to physical data structures, but more commonly to logical data structures.

It is from that point that coding begins. The information available to the coder should include, at a minimum, the input and output parameters for each independent program unit and an unambiguous description of the operations to be carried out by each. Analysis of information flow, performance or space requirements, and similar considerations lead to the identification of commonly used routines and data, yielding an initial program structure derived from the design.

A disciplined approach to software development, then, requires that the program *design stage* precede the program *construction stage*. The completed software design can be checked against the original specification by "walk-throughs" [3] or similar methods, with the resulting "software blueprints" providing the basis for implementation (or possibly redesign).

An important consideration in the target programming language, then, is the ease with which one can proceed from the design representation, with its modular structure and its degree of abstraction, to the program representation, i.e., executable code. A second key consideration is the ease with which one can determine the conformity between the completed program and the original specification, using testing and/or verification techniques.

## PLAIN AND ITS DESIGN CONTEXT

The past few years have witnessed an increased understanding of the relationship between programming languages and problem solving [4,5]. As a result of this work in programming methodology, programming languages are no longer viewed as independent entities, but rather as an integral part of the problem-solving process. Programming languages are now seen as a mechanism for expressing a problem solution in a precise way for computer execution. As such, a given programming language may have a significant effect upon the ease with which the solution may be expressed. If the language does not easily support the abstractions used by the programmer in solving the problem, then the transformation from the problem solution to a correctly executing program will be complex, with the increased likelihood that errors will be introduced during this transformation process.

A number of new programming languages have been designed and/or implemented with a primary or secondary objective of promoting proper programming techniques [6,7,8,9,10,11,12]. In addition, some general criteria for language designs have been advanced [13,14,15,16]. Design of the programming language PLAIN (Programming Language for Interaction) has proceeded in parallel with these other efforts, commencing in 1975. Unlike the other languages, the intended application area for PLAIN is interactive information systems, typically programs whose end users will be application-knowledgeable and computer-naive. PLAIN is intended to provide the application programmer with a tool that supports the systematic construction of this class of programs. As such, it contains facilities for definition and use of relational data bases, modules for information hiding, string processing with a simple pattern-matching facility, and exception-handling, incorporated into a well-structured, Pascal-based language.

In this paper, however, we shall be concerned primarily with the support provided by PLAIN for concepts of systematic programming. We begin by presenting some goals that encourage a disciplined approach to software construction, commenting briefly on their contribution to the overall goals. Then, following a short survey of other languages, we examine PLAIN with respect to these design goals, partic-

\* This work was supported in part by National Science Foundation grant MCS78-26287. Computing support for text preparation was provided by National Institutes of Health Grant RR-1081 to the University of California, San Francisco, Computer Graphics Laboratory, Principal Investigator: Robert Langridge.



ularly those of abstraction and modularity, and compare the approach of PLAIN with those of some other modern languages. Information on other aspects of the language and its implementation may be found in [11,17,18].

## LANGUAGE DESIGN GOALS FOR SYSTEMATIC PROGRAMMING

Although the intended application areas and the relative priority of the goals vary considerably among the recently designed languages, there are a number of areas of general agreement that can be identified. These common objectives, taken together, provide a sound basis for programming language design. Languages that meet these objectives can be expected to provide an excellent framework for the systematic construction of high quality programs. These objectives are presented briefly and with only the most significant aspects of their rationale, as additional discussion of these issues may be found in the cited references.

### 1) Support for abstraction

Abstraction has been recognized as a means to develop a representation of concepts that relates closely to the application being programmed, to hide inessential details of the problem solution at various levels of the program development process, and to support the notion of "top-down" design. If a problem solution involves the use of queues or directed graphs, for example, one should be able to make use of those objects in the programming process.

The ability to define these abstract objects, along with appropriate operations on these objects, is extremely valuable. Such objects can be specified formally using algebraic techniques to define their behavior [19]. If the objects and their associated operators are *encapsulated* so that the representation of the object is isolated and inaccessible from other parts of the program, the facility for *data abstraction* is analogous to the facility for *procedural abstraction* provided by functions and procedures in many programming languages.

Such a programming language facility, generically termed *abstract data types* [20], provides the programmer with the opportunity to define behavioral characteristics of data objects and to refine program and data structures in parallel. It is then possible to create data objects within a program resembling those used in the problem solution, thereby easing the process of transforming the problem solution into a program.

### 2) Support for modularity

Although there are a number of different definitions of a "module," for purposes of this paper, one may consider a module to be an object, perhaps a procedure, function, or abstract data type, that carries out a well-defined operation, hides a design decision, or isolates information from other modules. Typically, the actions may be described in a sen-

tence or two of natural language. Furthermore, each module has well-defined interfaces to other modules. Modularity makes an important contribution to the overall comprehensibility of programs, to the practice of programming by levels of abstraction, and to the production of large software systems by allowing various pieces of a software system to be effectively isolated from one another [21,22,23].

The ability to decompose a large problem into a number of smaller ones and to delineate clearly the interactions among the pieces is an important tool in gaining intellectual mastery over complex problems. Software design aids such as HIPO charts [24] and structure charts [25] have been developed to help identify modules and to represent the total structure of the software system so that the decomposed modules can be integrated into a single integrated system. Furthermore, concepts of cohesion (unity of function) and coupling (module connections) [22,25] provide a basis for evaluating module designs.

### 3) Support for verification and testing

Program correctness, as determined through either formal verification or testing, has been a critical motivation for much of the work in software engineering and programming language design. Verification is a formal mathematically-based proof that a program conforms to its specification. Testing is a collection of activities that provides a practical demonstration of conformity between the program and its specification, based upon systematic selection of test cases and execution of program paths and segments.

Both the characteristics of a given programming language and the practices used to write programs in the language affect verification and testing. The ease of testing and verification is further influenced both by *static* and *dynamic* program characteristics [26]. Static factors are those features that may be automatically checked by a compiler at translation time, those that are independent of the execution characteristics of the program. Examples of static aspects include most type checking and some checking for the use of aliasing.

Dynamic factors are those aspects of the program that are dependent upon its execution properties, including control flow and response to exceptional conditions. Issues of programming style, such as the use of uncontrolled branches and pointer structures, clearly affect the complexity of checking required.

Support for verification and testing is closely tied to some of the other issues as well. For example, the desirability of testing or proving program modules individually fits in well with the desirability of system design at the module level. In addition, support for verification and testing implies the prior development of system specifications and hence a systematic approach to software creation. Finally, other issues such as modularity and readability are closely related to issues of program correctness, since the determination of correctness is greatly aided by module simplicity and comprehensibility.

#### 4) Program readability

Program readability has been seen to be a valuable program property contributing to ease of program maintenance and modification [13]. The use of opaque programming “tricks” or the construction of cryptic programs is no longer considered to be an acceptable programming practice, as it has become recognized that programs must be read by humans as well as by machines during their increasingly long lifetimes.

Many properties combine to yield readable programs, including the use of mnemonic variable names, the presence of meaningful keywords, the liberal insertion of comments, and linear flow of program control. Here, too, programming practices are important, since it is possible to write a well-structured, highly understandable program in “poor” languages and a totally incomprehensible program in even the “best” language. Furthermore, program readability appears to be a highly personal and highly subjective quality, significantly influenced by the reader’s previous programming experience and programming style.

#### 5) Prevention of self-modifying programs

A number of languages, most notably LISP, treat programs and data interchangeably, in such a way as to permit the code being executed to vary dynamically, i.e., to be determined at execution time. Such an approach is entirely consistent with the concepts of stored programs and Von Neumann machines; unfortunately, though, this approach is in conflict with the goals of program readability and support for verification and testing, since the ability to create new variables and to alter the program dynamically makes verification and testing impossible unless one is able to test or prove all of the programs that can be generated. Furthermore, such programs are often difficult to comprehend, since the actual code is not totally visible. In Pascal and its descendants, procedures and data are separate entities, where data objects may change their values dynamically and procedures are static and immutable. Programs that permit “the execution of data” are forbidden.

#### 6) Control of scope and binding of variables

Block-structured languages provide explicit control over the existence of variables. Space for declared variables is allocated upon entry to a block and deallocated (except for statically allocated variables) upon exit from that block. The set of known variables can be determined from observing the static structure of the program, with no ability to create variables dynamically.

Control of the scope and binding of variables has been identified as a technique that can reduce programming errors caused by side effects, particularly those resulting from indiscriminate use of global variables [27]. Such control is also needed to achieve modularity, since, without it, a programmer may easily circumvent restrictions concerning the proper use of input and output parameters for a module.

The use of pointers should also be noted here, since they may contribute to this problem. Many languages, such as PL/I and Pascal, permit the creation of “dangling references” by having an object in an outer block point to an object in an inner block. When control leaves the inner block, the object pointed to may disappear, but the pointer itself will remain.

#### 7) Language size

Language size has also been seen to be important, since relatively small languages are easier to implement and can make it possible for the programmer to gain complete mastery of the programming language [13,14]. A number of different, albeit “rough,” metrics can be used to estimate language size, including the number of keywords, the size of its grammar (in LALR form, for example), the number of statement types, or the size of the compiler or interpreter for a given computer.

There appears to be an optimal size for languages, with some languages being so small as to prohibit an adequate variety of control structures or data types, while other languages are so large as to prevent the average programmer from gaining a clear understanding of the entire language, with all of its syntactic and semantic subtleties.

These seven design objectives are not orthogonal. Indeed, there are numerous intricate connections among them, as well as some inherent conflicts. For example, control of scope and binding of variables is closely related to modularity. On the other hand, restrictions on language size may serve to limit the extent to which a language may support a variety of abstractions. Thus, the language designer seeking to achieve these design objectives must give higher priority to some objectives than to others and must trade off various alternatives judiciously.

## LANGUAGES DESIGNED FOR SYSTEMATIC PROGRAMMING

As noted above, a number of different programming languages, including Pascal, CLU, Alphard, Gypsy, Euclid, LIS, PLAIN, Mesa, and Ada, have been designed with most or all of these design objectives in mind. (See [28] for example.) Even though the different languages are intended to serve a diversity of language requirements and applications areas, the languages have more similarities than differences when examined from the standpoint of support for systematic programming.

The most significant differences are those caused by different emphases in the design goals among the various languages. For example, Alphard and Euclid place a heavy stress on the goal of program verification, while the others might be said to *recognize* the importance of verification without the explicit requirement that programs in those languages *will* be verified. As another example, LIS and Euclid are seen as system implementation languages, to be used

primarily for the development of operating systems, compilers, and similar programs, while CLU and PLAIN are application languages. (This is not to imply that the languages in one group *cannot* be used for other applications, but only the intent of their designers.)

In the remainder of this paper, we will examine the design decisions in PLAIN with respect to these objectives for supporting a systematic approach to program construction, assessing some of the decisions in comparison and contrast with those made for other programming languages. The intent of this discussion is to provide some insight into the design of PLAIN and into some of the tradeoffs that were made in that design; the reader is not expected to agree with all of these decisions—if there were unanimous agreement on these issues, there would not be so many languages! In short, one of the implicit goals of many of these new languages (as can be seen from their defining documents) is to gain additional understanding of programming methodology and the ways in which language features aid or hinder the programming process.

From a software engineering standpoint, each may be regarded as a tool that can be made available to the individual software development group as an instrument for building their product. It is to be expected that some of these tools will receive little use and little acceptance, while the use of others will be strongly encouraged and modified and/or enhanced over time.

Finally, it should be noted that the programming language is part of a complete problem-solving process, which is supported by a software development methodology and a programming environment. The environment and the methodology will vary among organizations and among languages, but it is really the programming language, in combination with the programming environment, that determines the full extent of support for systematic programming that is provided for the programmer.

#### PLAIN: A LANGUAGE DESIGNED FOR RELIABLE INTERACTIVE SOFTWARE

As noted above, PLAIN (Programming Language for Interaction) is addressed to the dual goals of support for the construction of interactive programs, i.e., those programs that execute interactively and support for structured programming (in the original sense of that term [4]). PLAIN was designed with features to assist the development of programs involving conversational access to a data base.

These features include:

- 1) the data type **string** for variable length strings, along with appropriate operators and functions for string manipulation;
- 2) an elementary pattern specification facility along with pattern-matching operations, used both for validating user input and for formatting of input and output;
- 3) the data type **relation** and a set of operations to provide a facility for relational data base management [17,29];
- 4) a procedure-oriented exception-handling mechanism

for trapping errors and restricting control flow upon the occurrence of an exception, commonly used in the event of user input errors.

This set of features is largely missing from other programming languages that seek to support systematic programming. At the same time, those languages that are most heavily used for the construction of interactive program—BASIC, MUMPS [30], APL, LISP, and FORTRAN—are quite weak in meeting the design objectives stated above. PLAIN, by contrast, addresses both groups of design objectives.

From the outset, the original contribution of PLAIN was seen to be not so much the introduction of *new* language features, but rather a synthesis of features whose *interaction* would lead to a useful tool. In particular, the combination of relational data base management and facilities for data abstraction provides a powerful mechanism for structuring operations on data bases. Indeed, the design effort was undertaken with some reluctance, and only after a careful look at a number of other programming languages.

Given the planned number of innovations for supporting interactive programs, it was decided to be fairly conservative with respect to the inclusion of new features for systematic programming. The original intent was to remain fairly close to Pascal for these features; however, parallel developments in other language design efforts, including all of those mentioned above, were highly influential and the resulting language resembles Pascal somewhat less than was originally planned.

These new features are not only intended to support the creation of well-structured programs, but to go beyond that point so as to make a well-disciplined approach to program development a necessity for proper use of the language. In particular, it was considered extremely important to include features that aided modular decomposition of systems, with emphasis on intermodule communication [31], and to support joint refinement of procedures and data.

We now outline some features and design concepts of PLAIN that provide good problem-solving support and that impose various programming restrictions. The primary objective is not so much to present the PLAIN language in detail as to show the motivations of the design from the standpoint of programming discipline, with reference to the set of design objectives discussed above. Because of the interactions among these objectives, though, the subsequent discussion is structured along slightly different lines.

#### *Abstraction and modularity in PLAIN*

Abstraction and modular decomposition are two critical intellectual tools used by humans to solve problems. They are intricately related to one another, as each is intended to exhibit a *view* of a process or an object. For example, merely describing (at some level of abstraction) a process for sorting numbers into ascending order is inadequate for incorporating that process into a computer program; it is also essential to include a description of the interfaces between that operation and the host program.

To look at it another way, a module is a “black box” that provides an abstract view of a process or object to its invoker. Even though support for abstraction and support for modularity are presented as two separate design objectives, the extent to which one is achieved strongly affects the extent to which the other can be achieved. This is apparent if one considers the effect of being able to examine the internal structure of one module from another module; if one makes use of that internal information, then the abstraction is violated.

Many of the differences between Pascal and PLAIN are caused by the desire to provide better support for abstraction and modularity in PLAIN. Pascal has four key discernible weaknesses in this regard:

- 1) Unrestricted access to global variables—program units may freely access and/or modify variables declared in a containing lexical scope (unless the inner scope has a newly declared variable with the same name); thus, the use of specific variables is hidden, and a considerable amount of code inspection is required to determine the data flow. Access to dynamic structures via globally-declared pointers also makes it possible to create “dangling references,” since the object being pointed to may be deallocated.
- 2) Absence of input/output parameters for modules—parameters in Pascal are passed by value and by reference (**var**). However, passing a variable by reference is no guarantee that it is an output parameter, since it is considered a good programming practice (and an efficient one) to pass structured variables by reference, thereby eliminating the space and time required to make a copy of the parameter. Nonetheless, neither the procedure heading nor the procedure call gives an indication as to input or output parameters. Indeed, the concept of passing parameters by value and by reference is an *implementation* concept rather than a *programming* concept.
- 3) Lack of support for data abstraction modules—Pascal supports procedural abstractions (procedures and functions), but has no facility for defining encapsulated data types, similar to those present in CLU (a cluster), Alphas (a form), Euclid (a module), or others.
- 4) Side effects in functions—it is possible for a Pascal function to accept parameters by reference and to modify them within the body of the function; similarly, it is permissible for a function to make an assignment to a global variable. Such a capability goes against the mathematical concept of a function, as well as breaking down the abstraction embodied in the function and (effectively) creating additional output parameters from the function module.

PLAIN attempts to overcome each of these weaknesses, thereby providing stronger support for abstraction and modularity. First, all use of global variables must be declared in the heading of the individual program unit (procedure, function, data abstraction module). The PLAIN **imports** list is

similar to that of Euclid and the **glocon/glovar** declarations used by Dijkstra [32]. Some of these names are local declarations, some are parameters, but the rest are global variables or other program units. These nonlocal names must appear in the import list, along with a classification of their use, as **modified**, **readonly**, or **invoked**. This requirement does not apply to constants or to type declarations, which may be used freely. The effect of the imports list, though, is to increase the visibility of the use of variables throughout a program and to permit the reader of a module to determine the interrelationships between modules, both invocations and data connections.

In conjunction with use of the imports list to specify access to variables and program units, PLAIN contains the ability to restrict the use of a given variable to a designated set of program units. This feature, called the **restricted to** clause, controls the extent to which globally-declared variables may be used. With the imports clause alone, any global variable may be freely imported. However, there are many instances when it is desired to share a variable among a set of program units and to prevent it from being accessed by other units. (Labeled **COMMON** in FORTRAN can serve this same purpose.)

Consider, for example, a program in which routine main may call procedures *p1*, *p2*, and *p3*. Further, assume that *p2* and *p3* will both need the variable *k*, but that neither of them calls the other. Hence, communication of the value of *k* must occur through main. It is desired to prevent *p1* from obtaining (and possibly modifying) *k*. Thus, one can declare

```
var k: integer restricted to p2, p3;
```

as a way of achieving the desired protection.

Furthermore, PLAIN, like Ada, overcomes the dangling reference problem by forbidding deallocation of dynamically allocated variables. While this is not an entirely satisfactory solution from the standpoint of storage utilization, it is the only solution that permits the use of pointers without resorting to garbage collection and without permitting dangling references. The use of objects of pointer type is restricted in PLAIN in order to limit the number of program units that are aware of the representation of dynamically allocated objects.

Next, PLAIN has different rules from Pascal concerning parameters. PLAIN parameters may be either **readonly** or **modified**. A **readonly** parameter is an input parameter to the procedure or function whose value is not changed by the procedure or function. A **modified** parameter is a parameter that may have a value assigned to it during the execution of a procedure (possibly as a result of a call to a procedure invoked from within that procedure); as such the actual parameter for a formal modified parameter must be a variable. It may or may not have an input value. (An alternative strategy would have been to follow LIS and Ada, which have *in*, *out*, and *inout* parameters. The **readonly** parameters and the **modified** parameters are separated, in both the procedure declaration and the procedure invocation by the symbol “→”.)

For example, one might declare a procedure for the greatest common denominator with the following heading:

```
procedure gcd (m,n: integer→x,y,z: integer);
```

with a valid call appearing as

```
gcd (59,93→x,y,z)
```

where  $x$ ,  $y$ , and  $z$  have been declared as integers in the invoking routine.

This decision has several implications for implementation. First, conformity to the declaration must be checked to make sure that no assignment is made to readonly parameters. This involves making sure that the formal parameter does not appear on the left hand side of an assignment statement, in the modified part of an actual parameter list for a procedure called from within the given program unit, or as a modified variable imported into a lexically nested program unit. Although all of these checks can be made prior to execution time, they can involve a considerable amount of overhead.

An implementation advantage, however, is that it then becomes unnecessary to pass any of the parameters by value, thereby eliminating the overhead associated with copying of parameters. Because the use of the parameter can be checked from the program text, it is possible to pass all parameters by reference, regardless of whether they are readonly or modified. Thus, the programmer may accurately characterize all parameters as readonly or modified, depending upon their actual use. The overhead occurs at translation time and not during program execution.

The features described to this point have a significant impact upon the ease of transformation between the design phase and the program. Suppose that a system had been designed using the practices of Structured Design [19]. Part of the design representation is a structure chart showing the hierarchical structure of the system and the calls between modules. Each path between modules is numbered and an accompanying parameter table shows the input and output parameters for each module. For example, in Figure 1, the call to A2 from A (path 5) provides  $Y$  as an input parameter and obtains  $Z$  as an output parameter; it can be seen that  $Z$  is then passed to MAIN as an output of A (path 1).

Third, PLAIN contains a facility for encapsulation, bearing some resemblance to similar features in CLU, Euclid, and Ada. In addition to defining new types, one can also encapsulate a set of related procedures and functions, providing a feature similar to that of the Ada **package**. Each encapsulated type declaration consists of a **rep** clause, in which the representation of the type is declared, an **ops** clause, in which the operators upon the type are declared, an **exports** clause, in which the names of externally visible operators are given, and an optional **exception** clause, in which one can name exceptions associated with the operations upon the type.

The procedures read and write may be defined in the type to extend the built-in **read** and **write** operations. The Boolean function **equal** may be defined to extend the built-in **equal** function for structured variables. The procedure **init** may be

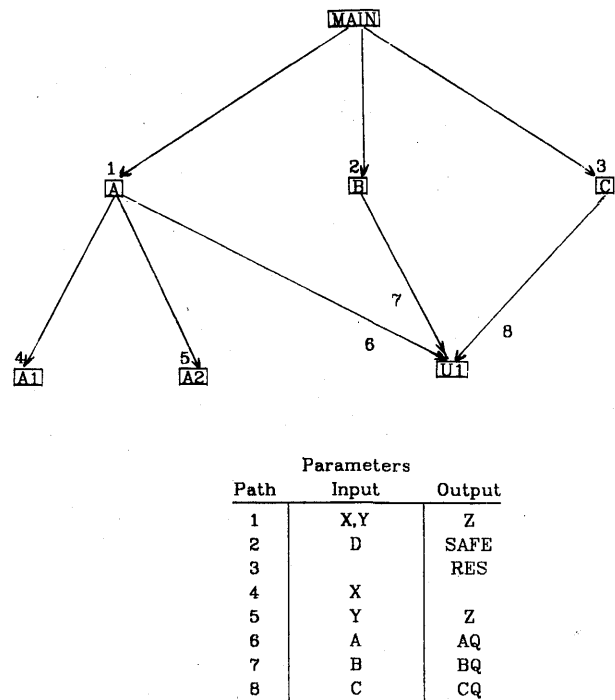


Figure 1—A structure chart

defined to specify actions to be carried out when a variable of that type is declared. The abstract type facility, along with several of the features previously described, can be illustrated by the familiar example of an integer stack.

The operations upon the stack may be specified as follows:

```
create:                →stack
push:   stack × integer →stack U stackfull
pop:    stack           →stack U stackempty
top:    stack           →integer U stackempty
empty:  stack           →Boolean
equal:  stack × stack   →Boolean
size:   stack           →integer
```

Axioms:

```
top(push(s,i))=i
top(create)=stackempty
pop(push(s,i))=if size(s)<MAX then s else stackfull
pop(create)=stackempty
equal(s1,s2)=if empty(s1) & empty (s2) then TRUE else
if empty (s1) | empty (s2) then FALSE else
(top(s1)=top(s2)) & equal (pop(s1),pop(s2))
size(create)=0
size(push(s,i))=size(s)+1
size(pop(s))=size(s)-1
```

Before presenting the PLAIN module, it is important to make some observations about the specification. First, the create operation is carried out by the declaration of a variable of the type **integerstack** in the program using the data ab-

straction. Thus, there is no explicit create operation in the integerstack module. Next, the stack specification given here is somewhat different from the specification given elsewhere in the literature [19,33], primarily to accommodate the stack-full result caused by the finiteness of machine resources.

The code for the module is shown in Figure 2. It should be noted that the implementation is not a *direct* encoding of the specification (hinting at some problems that verifiers might have). The primary difference is that the specification of equal uses a recursive definition, while the implementation examines individual elements of the stack. There are three reasons for this change: 1) recursion is usually more expensive in terms of machine resources; 2) pop is a procedure, not a function, and so cannot be used in the language in the way that it is used in the specification, and; 3) naming rules complicate the means of referring to individual objects in each of two different stacks being compared. In addition, one would have to make copies of the stacks to use a recursive equal operation without destroying the stacks; that, too, is more expensive than a simple element-by-element comparison.

Limited parameterization of the type definition is permitted, as shown by the stack size parameter MAX. The formal parameters must be of a simple type. Thus, one can use a single data abstraction to define integer stacks of different sizes, but not to define a stack of integers and a stack of

strings. The reason for this restriction is that relation is a data type and it was desired to prevent abstract type definitions from accepting relation as a parameter; the cleanest solution was a complete prohibition of type parameters. The resulting facility is less powerful (but easier to implement) than the **generic package** facility of Ada. One can now declare, for example,

```
var s1: integerstack [50]; s2: integerstack [100].
```

As noted above, it is the intention of PLAIN to disallow side effects in functions. At the simplest level, it is possible to make certain that no globals are imported and modified, and that no readonly globals or parameters are used as modified parameters in procedures called from within the function. Also, the syntax of the language forbids the presence of modified parameters; in their absence, it is impossible to use aliasing to cause side effects.

In order to be *strict* about the side effects requirement, though, more checking is required. First, certain data base operations must be prohibited; specifically, those modifying the current tuple indicator or the data base itself, caused by iterating through a relation, can be considered a side effect. Second, input/output operations must be restricted, since alterations to a file may be considered a side effect, especially if the file can be read after termination of the function. Such a restriction can cause complications for the software developer desiring to place debugging messages within functions, for example. Third, since functions may call procedures, all of the procedures called during execution of a function (to an arbitrary number of levels of invocation) would have to be checked to make certain that they, too, do not violate these restrictions on side effects.

In short, even though it is highly desirable to prevent *all* side effects, the costs of doing so, both in execution overhead and programmer inconvenience, must be considered. The prevention of input/output operations is particularly problematical in this regard, and PLAIN relaxes the side effect restriction to permit input/output within the body of functions. Otherwise, PLAIN requires sufficient declarations by the programmer in the heading of each program unit that it is possible to check procedures to see if assignments to global variables are made.

From an implementation standpoint, it is straightforward to check the restrictions on the use of globals. A flag can be set to indicate whether or not the stack of activations includes a function call. If there is an active function call, i.e., the calling sequence of program units includes a function, then the procedure to be executed must be checked for modified globals. Otherwise, the call is disallowed and an exceptional condition is raised. Note, though, that this is only a partial solution to the problem, since the declaration (in an imports statement) that a procedure can modify a global variable does not necessarily mean that the global *is* modified on a particular call to the procedure, since control flow may bypass any statements causing a disallowed assignment. Without this compromise, however, it would be necessary to check *every* assignment within such proce-

```

type integerstack [MAX: integer] =
  module
    exports push, pop, top, empty, equal;
    exception stackfull, stackempty;
    rep
      record
        stktop: 0..MAX;
        elements: array [1..MAX] of integer
      end record;
    ops
      function size(s:integerstack): integer; {computes size of stack s}
      begin
        size := s.stktop
      end size;
      function empty (s:integerstack): boolean; {returns true iff stack s empty}
      imports size: invoked;
      begin
        empty := size(s)=0
      end empty;
      procedure push (val: integer -> s: integerstack); {pushes integer val onto stack s}
      exception stackfull;
      imports size: invoked;
      begin
        if size(s) >= MAX then signal stackfull
        else s.stktop := s.stktop + 1; s.elements [s.stktop] := val end if
      end push;
      procedure pop (-> s: integerstack); {pops off top element of stack s}
      exception stackempty;
      imports empty: invoked;
      begin
        if empty(s) then signal stackempty else s.stktop := s.stktop - 1 end if
      end pop;
      function top (s: integerstack): integer; {returns value on top of stack; no pop}
      exception stackempty;
      imports empty: invoked;
      begin
        if empty(s) then signal stackempty else top := s.elements [s.stktop] end if
      end top;
      function equal (s1,s2: integerstack): boolean; {returns true iff s1 = s2}
      imports size: invoked;
      var i: integer;
      begin
        if size(s1) /= size(s2)
        then equal := false
        else
          i := size(s1); equal := true;
          loop
            if i=0 then exit end if
            if s1.elements[i] = s2.elements[i]
            then i := i - 1 else equal := false; exit
            endif
          repeat
            end if
          end equal;
          s.stktop := 0
        end integerstack;

```

Figure 2—Encapsulated type definition for integer stacks in PLAIN

dures, and the overhead of making those checks would be enormous. In summary, the seemingly innocuous desire to prohibit side effects in functions can impose severe restrictions and execution overhead.

PLAIN, then, provides considerable support for abstraction and modularity, providing additional features beyond those of Pascal at some expense in language size and complexity. The provision for abstract data types and the strong requirements for module interfaces enhance the possibility of creating libraries of procedures, functions, and encapsulated type definitions that can be used as "software components" [34].

### *Support for verification*

The design of PLAIN was motivated primarily by application needs; in the application areas addressed by PLAIN, there is a strong need for software and data reliability, particularly in areas such as medicine, where proper operation of a system may have life-critical importance. At the same time, though, the need for operational systems is so great that most developers of such systems tend to begin by writing code rather than by following any kind of coherent system design methodology. At present, there is almost no likelihood that anyone would attempt to prove the correctness of such a system, even had they produced a sufficiently rigorous specification.

Thus, support for program verification was not a major objective in the design of PLAIN, in the sense that it is in Alphard or Euclid. The assistance that PLAIN provides for program verification comes primarily through its resemblance to Pascal and to other modern languages. For example, PLAIN contains an `assert` statement that can be checked at execution time, but the statement only permits a Boolean expression, with no provision for such essential features as expressions involving universal or existential quantification. (Such quantification could be checked in a Boolean function that is part of the assertion.)

Along the same line, PLAIN is like Pascal with respect to aliasing, rather than including the features of Euclid that prevent aliasing. However, PLAIN improves upon Pascal with respect to the use of procedures and functions as parameters by requiring type information to be provided for the parameters of the procedure and function parameters. In this respect, it follows the proposal of the British Standards Institute for Pascal [35]. In this way, it is possible to perform a greater degree of type checking while still permitting function and procedure parameters.

This is not to say that the design of PLAIN ignores the possibility of verification, though, only that it was not a principal goal. A significant problem is that effective verification techniques have not yet been developed for the class of programs addressed by PLAIN. For example, very little has been done concerning verification of data base operations. Furthermore, even though the data base operations may be mechanically correct, it is impossible to guarantee with the present collection of facilities that the results are semantically meaningful.

PLAIN takes one small step in this regard, however, through its rules concerning type compatibility. In PLAIN, any two types having different names are different types. (The designers of Ada subsequently made the same definition.) Among the data base operations, the join operation of the relational algebra can only be performed on two objects of the same type. Thus, one can make judicious use of the data type facilities to assure that only meaningful joins can be performed.

As an example, consider two relations A and B, where A contains the attribute "age" and B contains the attribute "quantity^on^hand." If these attributes are both declared to be of type integer, then the relations A and B may be joined on these compatible attributes, however meaningless the result may be. If data types "agetype" and "amounttype" are defined in advance, though, with "age" declared to be of type "agetype" and "quantity^on^hand" declared to be of type "amounttype," then it becomes impossible to perform the join. In this manner, one may specify exactly which joins may occur and may verify their correctness from a logical standpoint.

Another verification problem is presented by the exception-handling mechanism. Once again, there are no practical methods for verifying programs in the presence of exceptional conditions; one might say that the occurrence of such a condition means that a program has failed to satisfy some input assertion and that the program therefore cannot be proved correct. Yet exception-handling is fundamental to PLAIN, since it is necessary to provide the programmer with facilities to prevent exceptional conditions from causing a program to terminate abnormally. The anticipated end users of PLAIN programs, being largely computer-naive, can be expected to make numerous errors, particularly in input, that must be properly trapped and handled; one simply cannot say that the program has failed to meet some input assertion and must therefore be terminated. Accordingly, the application programmers writing programs in PLAIN must be given the ability to trap and handle exceptions.

The PLAIN exception-handling mechanism, described at length in another paper [36], seeks to provide a well-structured flow of control following the occurrence of an exceptional condition. The programmer may create a **handler procedure** that can be associated with the occurrence of a specific exception at a specific program location. When an exception is raised, either through the `signal` statement, or through an automatic mechanism in the language processor, the handler procedure can carry out any required actions, potentially clear the offending exception, and then return control to normal program flow, to the beginning of the statement in which the exception occurred (`retry`) or to the invocation point of the procedure in which the exception occurred. In this way, exceptions can be passed through succeeding levels of invocation with any necessary actions being taken at each level. Since exception-handling is done with procedures, it is possible to pass parameters from the environment of the exception to the handler procedure, following the normal rules for scoping of declarations. At any point, the active exception may be cleared by the handler for that level so that normal program operation can continue.

The intent of this approach is to facilitate both the programming of exception-handling actions and the verification of programs in the presence of exceptions, since this method avoids the unrestrained flows of control and unrestricted access to variables that characterize some of the other exception-handling schemes. Although a more detailed approach to this verification is sketched out in [36], there has not yet been any practical experience with the application of verification techniques to such programs.

#### *Support for program readability*

Although, as previously noted, program readability is difficult to quantify and can be strongly affected by individual programming styles, it is possible to provide language features that enhance program comprehensibility. Many of these features provide support for other systematic programming goals as well. In general, the design of PLAIN attempts to follow Hoare's dictum that "the readability of programs is immeasurably more important than their writability" [13].

As with many other language aspects, much of the readability of PLAIN programs results from its resemblance to Pascal. Among the common features supporting readability are:

- provision of appropriate keywords
- format free program structure permitting indentation on lines
- control structures supporting linear flow of program control within program units
- prevention of self-modifying programs
- straightforward provision for comments
- limited language size.

Similarly, the Pascal-like program structure retains the disadvantage of placing the main program at the end of the program text.

PLAIN incorporates some additional features intended to enhance program readability (as well as to help in achieving other goals). These features are the following:

- fully bracketed control structures
- explicit importing of global names into a module
- input/output parameter lists in both declaration and call of procedures.

The use of fully bracketed control structures permits a more consistent language definition and can reduce the use of **begin-end** pairs as separators. The reduction in **begin-end** pairs not only eliminates unnecessary program "clutter," but also removes a major source of programming errors, making the **begin-end** now serve only the single purpose of enclosing an entire executable program unit (main program, function, or procedure).

In Pascal, for example, the structure of the **if** statement is

```
if Booleanexpression then statement [else statement].
```

In PLAIN, as well as in Ada and other newer languages, it is

```
if Booleanexpression then statementlist
[else statementlist] end if.
```

Similar gains are achieved with the **case** statement. The statement is terminated with an **end case** and individual cases are separated with the reserved word **when**. Again there can be a considerable reduction in the number of **begin-end** pairs, producing a situation in which both readability and writability are improved.

The imports list, discussed above, in addition to helping enforce rules concerning modularity, is an aid to program readability. Because declarations and imported names are all visible in the heading of a program unit, it is easier to comprehend, modify, and/or validate units independently. The designers of Ada have taken the opposite view, claiming that importation of a large number of objects will *detract* from program readability and cause additional clutter. This author believes that the proper use of structured objects, combined with efforts to minimize coupling between modules, will prevent the imports list from becoming excessively long, and that its presence provides a good mechanism for specifying the interface between the PLAIN program and its execution environment. Further experience in the use of these languages may help to resolve this difference.

Another improvement to readability comes about from the restrictions on the use of pointer variables in PLAIN. Because pointer variable may only be used within modules, most program units are free of expressions involving complicated data access methods, such as multilevel pointer structures. While PLAIN does not achieve a uniform reference mechanism, the number of reference methods is quite small. Furthermore, function and procedure calls must be used to access the operations on the complex data structures defined in data abstractions. This restriction has several benefits:

- access to the physical representation of a data object is sharply restricted so that the reader of the program only needs to understand the logical operations on the object once the isolated representational information is understood
- the reader, typically performing a maintenance activity, needs to study much less of the program text in order to make changes to the data structures
- meaningful names can be chosen for the functions and procedures, thereby aiding reader understanding of the program.

It must also be recognized that some of these gains in readability come at the expense of some overhead in space



or execution time as a result of the additional procedure and function calls needed to accomplish the encapsulation of data.

## CONCLUSION

This paper has examined the design of the programming language PLAIN from the standpoint of the support that it provides for the notions of systematic programming, focusing on both its strengths and weaknesses. It can be seen that the design of PLAIN places major emphasis on the goals of abstraction, modularity, and readability, and that it makes advances over Pascal and features of some other modern languages with respect to supporting a well-disciplined approach to software construction.

At the same time, support for program verification and testing was consciously left at a lower level than is possible given the current technology of programming language design. The language size is moderate, containing more features and more syntax than Pascal, and being comparable to Ada in that respect. The goal of small language size was not achieved as fully as had been hoped, due to the apparent needs of the application area.

The implementation of PLAIN is presently under way on the PDP-11 computer under the UNIX operating system, and it is expected that an initial implementation will be operational in the summer of 1980. It is anticipated that implementation experience and increased use of the language will eventually lead to revisions in the language to provide improved support for the dual objectives of aiding the construction of interactive information systems and encouraging the use of systematic programming methodology.

## REFERENCES

- Peters, L. J. and Tripp, L. L., "Software Design Representation Schemes," *Proc. of the Symposium on Computer Software Engineering, MRI Symposium Proceedings*, vol. 24. Brooklyn: Polytechnic Press, 1976, pp. 31-56.
- Caine, S. and Gordon, E., "PDL—a Tool for Software Design," *Proc. AFIPS 1975 NCC*, vol. 44, pp. 271-276.
- Myers, G. J., "A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections," *CACM*, vol. 21, no. 9 (September, 1978), pp. 760-768.
- Dahl, O.-J., Dijkstra, E. W. and Hoare, C. A. R., *Structured Programming*. London: Academic Press, 1972.
- Wirth, N., "Program Development by Stepwise Refinement," *CACM*, vol. 14, no. 4 (April, 1971), pp. 221-227.
- Wirth, N., "The Programming Language Pascal," *Acta Informatica*, vol. 1, no. 1 (1971), pp. 35-63.
- Wulf, W. A. (ed.) *et al.*, "An Informal Description of Alphard" (preliminary), Department of Computer Science, Carnegie-Mellon University, February, 1978.
- Liskov, B., *et al.*, "CLU Reference Manual," MIT Laboratory for Computer Science, Computation Structures Group Memorandum 161, July 1978.
- Lampson, B. W. *et al.*, "Report on the Programming Language Euclid," *ACM SIGPLAN Notices*, vol. 12, no. 2 (February, 1977), pp. 1-79.
- Ambler, A. L. *et al.*, "Gypsy: a Language for Specification and Implementation of Verifiable Programs," *Proc. of ACM Conf. on Language Design for Reliable Software, ACM SIGPLAN Notices*, vol. 12, no. 3 (March, 1977), pp. 1-10.
- Wasserman, A. I. *et al.*, "Revised Report on the Programming Language PLAIN," Laboratory of Medical Information Science, University of California San Francisco, Technical Report #34, July, 1978. (Revised Report in preparation).
- Ichbiah, J. D. *et al.*, "Preliminary Ada Reference Manual," *ACM SIGPLAN Notices*, vol. 14, no. 6 (June, 1979), part A.
- Hoare, C. A. R., "Hints on Programming Language Design," Stanford University Computer Science Department Technical Report CS-73-403, December, 1973.
- Wirth, N., "On the Design of Programming Languages," *Information Processing 74*. Amsterdam: North-Holland, 1974, pp. 386-393.
- Richard, F. and Ledgard, H., "A Reminder for Language Designers," *ACM SIGPLAN Notices*, vol. 12, no. 12 (December, 1977), pp. 73-82.
- Department of Defense Advanced Research Projects Agency, "Requirements for High Order Computer Programming Languages—'STEEL-MAN'," June, 1978.
- Wasserman, A. I., "The Data Management Facilities of PLAIN," *Proc. ACM 1979 SIGMOD Conference*, 1979, pp. 60-70.
- Booster, T. W., "Implementation of Pattern Matching in PLAIN," M.S. Project Report, University of California, Berkeley, September, 1979.
- Guttag, J. V., "Abstract Data Types and the Development of Data Structures," *CACM*, vol. 20, no. 6 (June, 1977), pp. 396-404.
- Liskov, B. and Zilles, S. N., "Programming with Abstract Data Types," *ACM SIGPLAN Notices*, vol. 9, no. 4 (April, 1974), pp. 50-59.
- Parnas, D. L., "On the Criteria to be Used in Decomposing Systems into Modules," *CACM*, vol. 15, no. 12 (December, 1972), pp. 1053-1058.
- Myers, G. J. *Reliable Software through Composite Design*. New York: Petrocelli/Charter, 1975.
- Liskov, B., "A Design Methodology for Reliable Software Systems," *Proc. AFIPS 1972 FJCC*, vol. 41, pp. 191-199.
- HIPO—a Design Aid and Documentation Technique. White Plains: IBM Data Processing Division. Pub. GC20-1851.
- Yourdon, E. and Constantine, L. L., *Structured Design*. Englewood Cliffs, NJ: Prentice-Hall, 1979.
- Wasserman, A. I., "Testing and Verification Aspects of Pascal-like Languages," *Journal of Computer Languages*, vol. 4, no. 3/4 (1979), pp. 155-169.
- Wulf, W. A. and Shaw, M., "Global Variables Considered Harmful," *ACM SIGPLAN Notices*, vol. 8, no. 2 (February, 1973), pp. 28-32.
- Ichbiah, J. D. *et al.*, "Rationale for the Design of the Ada Programming Language," *ACM SIGPLAN Notices*, vol. 14, no. 6 (June, 1979), part B.
- Codd, E. F., "A Relational Model of Data for Shared Data Banks," *CACM*, vol. 13, no. 6 (June, 1970), pp. 377-387.
- American National Standards Institute. *MUMPS Language Standard*. ANSI X11.1-1977.
- DeRemer, F. and Kron, H., "Programming-in-the-Large vs. Programming-in-the-Small," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 2 (June, 1976), pp. 80-86.
- Dijkstra, E. W. *A Discipline of Programming*. Englewood Cliffs: Prentice-Hall, 1976.
- Wulf, W. A., London, R. L. and Shaw, M., "An Introduction to the Construction and Verification of Alphard Programs," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4 (December, 1976), pp. 253-264.
- Belady, L. A., "Evolved Software for the 80's," *Computer*, vol. 12, no. 2 (February, 1979), pp. 79-82.
- Ravenel, B., "Toward a Pascal Standard," *Computer*, vol. 12, no. 4 (April, 1979), pp. 68-82.
- Wasserman, A. I., "Design and Evaluation of a Procedure-Oriented Exception-Handling Mechanism," in preparation, 1980.

# Some practical experiences with the Pascal language

by G. G. GUSTAFSON, T. A. JOHNSON and G. S. KEY

*Computer Sciences Corporation*  
San Diego, California

## INTRODUCTION

In 1976 the Naval Ocean Systems Center (NOSC) initiated the BIODAB project to determine if a high performance relational DBMS could be developed to support scientific applications.

The BIODAB prototype was implemented in FORTRAN V and small percentage of Assembly Language (less than 10 percent). The modifications needed to enhance the static prototype for general purpose use involved the addition of several major processors, including: Data Base Definition, Update (Add, Change, and Delete), and Unload. In addition, the Report Writer had to be extensively modified for greater efficiency and flexibility.

The results of that study encouraged NOSC to develop RELATABASE as a general-purpose enhancement of the BIODAB prototype.<sup>1</sup> The design constraints for RELATABASE were threefold. First, retain the high performance data access technology developed during the BIODAB project. This constraint required a hardware architecture in which sequential searching of a highly compressed data base was practical. An access method employing double-buffered, asynchronous I/O with a hardware masked search instruction was developed for this purpose. Second, RELATABASE should not use any more main memory than necessary for each of its many functions. Finally, RELATABASE should be an interactive system capable of supporting ad hoc queries of a data base. All of these criteria were satisfied using the UNIVAC 1110 at NOSC.

We define the "production environment" to be that environment in which programs are developed as deliverable products under constraints of both time and money. It is our objective to distinguish organized programming of this type from systems, research, hobbyist, or other types of programming.

## THE DEVELOPMENT EFFORT

### *Implementation language choice*

The choice of the implementation language was carefully considered. Many software tools were available in FORTRAN from the BIODAB project, and all team members were competent FORTRAN programmers. A preprocessor for writing structured FORTRAN programs<sup>2</sup> was also avail-

able. In spite of these tools, Pascal was seriously considered as the new applications language for the following reasons: (1) as a block-structured language, Pascal did not require a preprocessor; (2) most of the tools which had been built for FORTRAN, and particularly for character manipulation, were either unnecessary or easily rebuilt in Pascal; (3) because Pascal-1100<sup>3</sup> allows direct access to the UNIVAC-1100 Executive Request mechanism, reliance on special Assembly Language routines could be significantly reduced; as an example, Figure 1 lists the RELATABASE external module DRUMIO which interfaces directly with the executive and performs random access I/O; the replacement of Assembly Language routines with ones written in a high level language is usually desirable because the code is easier to read and maintain, and because the maintenance function can usually be performed by less senior programmers; (4) although only one of the team members had previous experience with Pascal, learning the language was not expected to be a significant problem; (5) because the new software would rarely need to communicate with the existing FORTRAN code, the interface between the two languages would not be a problem; (6) the control structures, data structures, and data type security features of Pascal were definite advantages.

In order to promote rapid learning of Pascal and uniform coding practices, a set of programming standards was chosen and imposed on all Pascal code.<sup>4</sup> A similar set of standards<sup>5</sup> was provided for all new FORTRAN code which was needed to support existing BIODAB code. The Pascal standards included: descriptive preamble comments for each independently compiled procedure or function; the capitalization of all Pascal reserved words; a few indentation rules to make data and control structures more visible.

### *Project experience with Pascal*

The Pascal programming team had three members. Two full-time project members had experience in the production environment but did not know Pascal. The other programmer was a part-time member who had used Pascal extensively as a student.

The inexperienced Pascal programmers found that mastery of the language came slowly at first for a number of reasons. The most common problems were remembering to use the semicolon as a statement separator and the lack of

```

const
  iow = 3B ;
type
  ascidentype = packed array [1 .. 12] of CHAR ;
  fdidentype = packed array [1 .. 12] of 0 .. 77B ;
  halfword = 0 .. 777777B ;
  validcodes = 10B .. 53B ;
  packetype = packed record
    filename : packed array [1 .. 12] of 0 .. 77B ;
    wordtwo: INTEGER ; (* FILL *)
    status: 0 .. 77B ;
    iofunction: validcodes ;
    s3word3: 0 .. 77B ; (* FILL *)
    finalcount: halfword ;
    nwords: halfword ;
    bufferaddress: halfword ;
    diskaddress: INTEGER ;
  end ;
  (* ASCII to FIELDATA string conversion *)
  procedure asc2fd ( inputstring : ascidentype ;
    nrchars : INTEGER ;
    var outputstring : fdidentype ) ; EXTERN ;
  procedure ENTRY drumio ( filename : ascidentype ;
    iofunction : validcodes ;
    bufferaddress : halfword ;
    diskaddress : INTEGER ;
    var nwords : halfword ;
    var status : INTEGER ) ;
var
  iopacket : packetype ;
begin
  if nwords <= 0 then begin
    nwords := 0 ;
  end
  else begin
    asc2fd ( filename, 12, iopacket.filename ) ;
    iopacket.iofunction := iofunction ;
    iopacket.nwords := nwords ;
    iopacket.bufferaddress := bufferaddress ;
    iopacket.diskaddress := diskaddress ;
    iopacket.status := 0 ;
    a0 := ADDRESS ( iopacket ) ;
    ER ( iow, a0 ) ;
    status := iopacket.status ;
    nwords := iopacket.finalcount ;
  end ;
end ; (* drumio *) .

```

Figure 1—Pascal-1100 executive request (ER).

a construct-closing keyword which identified which structure was being closed (i.e., something like the END IF in ANSI X3.9-1978 FORTRAN<sup>6</sup>). The latter problem was solved with some satisfaction through indentation of nested constructs. However, once the learning pains were past, the power and flexibility of Pascal began to be appreciated. Code generally was written faster and easier than would have been possible in FORTRAN. Data type security proved to be a most helpful feature in reducing run time errors. The *record* and *set* (Figure 2) data structures were well received and saw frequent use. A record type is a most convenient way to group related data in a logical and clear manner. The ability to build and manipulate sets offers the programmer many new and powerful approaches to algorithm development.

Because the implementation of RELATABASE followed the top-down concept with functional independence of modules, we encountered no problems of module interfacing. Most of the debugging process was limited to minor logic errors in newly introduced modules. The modification of the standard procedure WRITELN in Pascal-1100 to allow numeric variables to be displayed in octal was a great help because several of the data types we used were defined as packed, partial computer words and could not be meaningfully displayed otherwise.

### Project productivity

The productivity of the project can be divided between the efforts of the FORTRAN team and the Pascal team. In

all, RELATABASE contains 53,600 lines of source code. Of that total, 9,100 lines were retained from the BIODAB program, 23,000 lines were new FORTRAN code (generated by the FORMEL 'structured FORTRAN' preprocessor), 3,200 lines were Assembly Language code to support Pascal programs, and 18,300 lines were written in Pascal. The implementation of RELATABASE began in mid-January with a five-man team, and continued through September of the same year. During that time the staff size varied, decreasing until only one member remained at the project's conclusion. The total number of man-hours given to the entire project was 4,574.

The production rates for Pascal and FORMEL code have been adjusted to account for source lines added because of the coding conventions (Pascal) or preprocessor (FORMEL) used. The adjustment factors are 90 percent of total lines for Pascal and 75 percent of total lines for FORMEL. On this basis, Pascal production rates were 50 lines per programmer-day. FORMEL production is computed at 86 lines per programmer-day. The BIODAB project, where 90 percent of the code was written in FORTRAN, without a preprocessor, produced 45 lines of source code per programmer-day. These data are summarized in Table I.

In order to compare the Pascal rate with FORMEL's, one must keep in mind that the FORMEL staff was composed of expert FORTRAN programmers, while two of the three members of the Pascal staff had never coded in Pascal be-

```

type
  delimitype = set of CHAR ; (* delimiters for scanner *)
  scannerrec = record (* for scanner control *)
    columnfound : INTEGER ; (* column delimiter found *)
    delimiterfound : CHAR ; (* stop scan delimiter *)
    firstcolumn : INTEGER ; (* non-ignore char column *)
    ignorechar : CHAR ; (* ignored leading char *)
    itemlength : INTEGER ; (* length of string found *)
    maxinpchars : 1 .. MAXINT ; (* chars to scan, usually 80 *)
    startcolumn : 1 .. MAXINT ; (* char pos to start scan *)
  end ;
  fullstring = packed array [1 .. 133] of CHAR ;
var
  column : 1 .. MAXINT ;
  match : BOOLEAN ;
  procedure ENTRY scanner ( input_string : fullstring ;
    delims : delimitype ;
    var controlvalues : scannerrec ) ;
begin (* scanner *)
  with controlvalues do begin
    if ( startcolumn <= maxinpchars ) then begin
      column := startcolumn ;
    end
    else begin
      column := 1 ;
    end
    columnfound := 0 ;
    firstcolumn := 0 ;
    itemlength := 0 ;
    while ( input_string [column] = ignorechar ) and
      ( column <= maxinpchars ) do begin
      column := column + 1 ;
    end ;
    if ( column <= maxinpchars ) then begin
      firstcolumn := column ;
    end ;
    match := FALSE ;
    while not match and ( column <= maxinpchars ) do begin
      match := input_string [column] in delims ;
      if match then begin
        delimiterfound := input_string [column] ;
        columnfound := column ;
      end
      else begin
        itemlength := itemlength + 1 ;
      end ;
      column := column + 1 ;
    end ;
    startcolumn := column ;
  end ;
end ; (* scanner *) .

```

Figure 2—Record and set usage.

TABLE I.—Summary of Language Productivity

Source	Hours	Source Lines		Productivity	
		Unweighted	Weighted	Unweighted	Weighted
PASCAL	2774	21500	17250	62	50
FORMEL	1800	23000	19350	102	86
ALL	4574	44500	36600	78	64

fore. Since then one of the Pascal programmers has written production programs at a rate of 300 lines of debugged code per day.

### DIFFICULTIES WITH PASCAL

Pascal offers significant advantages in terms of data and control structures, but exhibits some disadvantages in the production environment. These disadvantages may be due in large part to the design goals which Wirth chose for the language.<sup>7</sup> The current standardization effort for Pascal may eliminate most of these disadvantages. The production programmer deciding upon an implementation language should, however, be familiar with Pascal's disadvantages as well as its advantages.

We encountered two broad categories of problems using Pascal in a production environment. The first set of problems was associated with the language definition itself. The second set was associated with the specific language implementation that we used. In fairness, we hasten to add that a number of the difficulties may have arisen from the manner in which RELATABASE was developed using multiple source languages (i.e., FORTRAN, Pascal and Assembly).

The difficulties with Pascal as a production language can be further subdivided into those whose repair we consider to be mandatory and those whose repair we consider to be desirable. "Mandatory" implies that without some solution to the specified problem, preferably a new standard for the language, Pascal cannot provide the production programmer with needed facilities. "Desirable" implies that a solution should be sought within the language standard, although a solution, albeit awkward, can be devised by the programmer. Desirable also includes those facilities which are primarily a convenience or could be included within a particular implementation.

#### Language definition

##### Standard types

Failure to provide standard types equivalent to the FORTRAN types DOUBLE PRECISION and COMPLEX causes considerable extra programming with Pascal. Had Pascal remained a teaching language, then the need for these data

types would be reduced. It can be argued that Pascal supports the programming of double precision (or any precision) arithmetic operations.<sup>8</sup> However, the need to develop specialized arithmetic routines is not in keeping with an environment in which a customer is paying for a product. The same argument applies to the complex arithmetic operations. It may be easy to define complex variables through appropriate type statements, but coding and recoding the complex arithmetic operations through functions and procedures is undesirable in a production environment.

Because Pascal supports strong type checking, it is necessary to include a type-less operator akin to the intrinsic function `BOOL` in UNIVAC's FORTRAN V<sup>9</sup>:

```
(bool-stmt)      := BOOL((one-word-var))
(one-word-var)  := <integer-variable>
                := <real-variable>
                := <logical-variable>
                := <typeless-variable>
```

The result of the `BOOL` function is to wholly ignore any type incompatibilities which might arise during arithmetic or Boolean operations upon variables. The power of this function must be limited to logical expressions. Its use would be inappropriate in an assignment statement, for example.

#### External compilation

Two associated problems exist within a multi-language environment. Pascal does not define the mechanism needed for independent external compilation of modules. In large-scale software development, where more than one programmer is involved in development, it is mandatory that modules be developed independently and tied together through mechanisms offered by the operating system or system processors. Requiring five programmers to simultaneously edit the same workspace is justification enough for external compilation.

In complex systems it is frequently necessary to collect programs into separately addressable banks. This technique, known as "bank-named collection," minimizes core-second charges by switching out banks which are not currently required.<sup>10</sup> In RELATABASE, up to four active banks were resident in core, and occasionally the need arose to communicate between them. Pascal does not support this requirement, particularly if one of the two communicating banks is switched out.

FORTRAN solved the problem posed by switched bank communication through the `COMMON` statement. A named `COMMON` can be placed in a control bank (i.e., one that would not be switched out), and any FORTRAN module can make updates to it. This feature was particularly useful in maintaining the status of a bank that was executed earlier but is no longer active. Although RELATABASE designers were able to work around this problem, we believe the need for a Pascal equivalent to FORTRAN's `COMMON` is mandatory.

### Dynamic arrays

The bounds of an array are sometimes ill-defined in the production environment. Scientific programs in particular often use core as a workspace rather than as a specific pre-defined entity. Matrix manipulations may be vector oriented, but type incompatibilities arise when attempting to reference a portion of a matrix rather than the whole matrix. Likewise, it is sometimes necessary in systems-level programming to discard a number of words which are not meaningful to a particular process. Because Pascal does not allow direct referencing of partial arrays, the programmer is faced with the necessity of a type statement of the form:

```
type
  area = packed record
    case BOOLEAN of
      TRUE : ( record1 : ... ) ;
      FALSE : ( record2 : ... ) ;
    end ;
```

This construct affects the results of the FORTRAN EQUIVALENCE statement. Unless dynamic arrays are included in the language standard, we urge caution to those who advocate that the tag field be required in variant records.<sup>11</sup> Modifying Pascal to allow dynamic, or adjustable, arrays meets most of the production environment needs and therefore becomes mandatory.

### Parameterization of constants

Advocates of Pascal, ourselves included, frequently assert that Pascal programs are portable. This statement is not generally true because of the inability of the programmer to define constants, particularly parameterizing constants, in terms of previously declared constants. Constant definition parts often include annoyances such as:

```
const
  pi = 3.15159265 ;
  two-pi = 6.28318531 ;
```

rather than the statements:

```
const
  pi = 3.15159265 ;
  two-pi = 2.0 * pi ;
```

Nor is the code:

```
const
  high = 256 ;
  highm1 = 255 ;
  highp1 = 257 ;
```

conducive to either portability or reliability. The need to declare, within the constant part only, constants in terms of previously declared constants is mandatory. We believe that

the form of constant declaration should allow any type of constant expression—arithmetic, relational and multiplicative operations—upon previously declared constants.

### Data initialization

The failure to provide compile-time data initialization is a source of both increased costs and decreased reliability. This problem is particularly important when more than one programmer is involved in development of a system which requires large amounts of initialized data. Both factors make some form of compile-time initialization mandatory.

### Declare before use

The enforcement of the “declare before use” rule for procedure and function declarations, together with the requirement for the FORWARD directive, appears to be a design flaw. Even single-pass compilers should be able to recognize the failure to declare a module before the termination of compilation. We believe it desirable to remove the restrictions imposed by this rule with respect to procedure and function declarations.

### Termination of comments

One of the more annoying problems associated with debugging a Pascal program is caused by the failure to include (or the accidental removal of) the comment terminator (the Pascal symbols “}” or “\*)”). The compiler ignores the balance of the code and usually produces voluminous messages at the end of the listing. There are few errors which are more difficult to diagnose than failure to close a comment because the programmer frequently “sees” comments termination, even though none exists. Although some Pascal implementations warn of a possible error, usually when a semicolon is encountered, we believe the end of source line should act as a comment terminator. We admit that this need can be repaired only by a modification to the language.

### For-statement syntax

As an aid to producing structured code, the syntax of the for-statement could be modified. The recommended form would be:

```
“for” <loop-variable> “:=” <start-value> (“to” |
  “downto”) <stop-value> [“by” <step-value>]
  [“when” <Boolean-expression>] <statement>.
```

The failure to provide a means of stepping through values (i.e., the proposed by-clause) appears to be a major cause for error in production code. The contrivances needed to account for non-step-by-one values of the loop control-variable are counter to one of the more important design goals

of the language: "a systematic discipline based upon certain fundamental concepts clearly and naturally reflected by the language"<sup>7</sup> (pg. 133). Forcing recomputation of a pseudo loop variable within the body of a loop is not consistent with a natural or clear language.

An early escape mechanism is needed in the syntax of the for-statement. The proposed when-clause is a precondition to loop execution in the same manner as the usual precondition required of the current value of the loop control-variable. Therefore, the for-statement body will be executed if, and only if, the Boolean expression yields a value of TRUE and the value of the loop control-variable is within the range of start-value to stop-value, inclusive. We would not require that the declaration of the loop control-variable be external to the loop body. Because the variable's value is usually, and we believe correctly, undefined at termination of the loop, it seems an unnecessary requirement to declare the variable externally to the for-statement.

### Standard functions

The lack of standard functions in Pascal requires consideration. The functions provided for numerical analysis (e.g., ABS, ARCTAN, COS, EXP, LN, SIN, SQR, and SQRT) are simply not enough in the production environment. The FORTRAN intrinsic functions (e.g., MIN (choosing smallest value), MAX (choosing largest value), LOG10 (common logarithm), TAN, ASIN (arcsine), ACOS (arccosine), SINH (hyperbolic sine), COSH (hyperbolic cosine), and TANH (hyperbolic tangent)<sup>6</sup> are important in scientific programming. Furthermore, requiring these functions to be developed during a production project raises the specter of accuracy and precision errors. Because many computer systems contain these functions as a part of their vendor-supplied libraries, their repeated recoding is even less desirable. A cogent argument has been made for inclusion of additional operators in programming languages.<sup>12</sup> Additional operators in Pascal would be most welcome.

### Input/Output facilities

The I/O facilities of Pascal require major revision and redesign. We will mention only two of our difficulties. Data types, other than the standard types, cannot be displayed directly. A common solution is: define a variant record type, one variant of standard type and the other of the type to be displayed; assign to the type to be displayed variant of the record the value of the variable to be displayed; display the standard type variant. For example, if the current address contained in a pointer was to be displayed, the statements of Figure 3 might be used. The second difficulty with the Pascal I/O facilities is that data cannot be displayed in other than the standard type base. For example, an integer cannot be displayed as some power of eight (i.e., octal). These difficulties have a direct effect upon the work required to perform program debugging. Any improvements would be desirable.

```

type
  link = ↑node ;      (* pointer into plot data tree *)
  node = record      (* binary tree of plot data *)
    left : link ;    (* pointer to leftson *)
    right : link ;   (* pointer to rightson *)
    x_value : REAL ; (* value of x *)
    y_value : REAL ; (* value of f(x) *)
  end ;
  print_type = packed record
    case BOOLEAN of
      TRUE : ( pointer_rep : link ) ;
      FALSE : ( integer_rep : INTEGER ) ;
    end ;
var
  plot_points : link ; (* binary tree of plot values *)
  print_pointer : print_type ;
  :
  :
  :
  print_pointer.pointer_rep := plot_points ;
  WRITELN ( OUTPUT, print_pointer.integer_rep ) ;

```

Figure 3—Printing non-standard data types.

### Language implementations

The difficulties derived from implementation are spawned in part by the valid insistence that Pascal-1100 remain as close to the standard as possible. We regard these difficulties as dependent upon the implementation and do not consider it advisable to include them within any Pascal language standard.

### Source code inclusion

We found the need to copy source code into a module from another system-known entity to be a mandatory requirement for production implementations. This condition was especially true for type declarations which, in the Pascal-1100 environment, were required in both the calling and called modules (Pascal-1100 supports external compilation). The facility which meets this need is the COBOL COPY verb,<sup>13</sup> although with perhaps somewhat less of a baroque form. Unless a program development group has a powerful text editor, the failure to provide a form of source code inclusion can impact the development schedule.

### Identifier names

To the extent practical, it is desirable to have a Pascal implementation accept identifier names of up to the maximum length which can fit on a source line. If a compiler limits unique identifier names to, say, 12 characters, as does Pascal-1100, prefixes which are usually meaningless are bound to be attached to the name. These prefixes tend to reduce readability and increase maintenance costs.

### CONCLUSIONS

Pascal is a relatively new addition to the family of production programming languages. We believe that as the language matures Pascal will become accepted as a superior programming tool. The RELATABASE project has shown that the language is easily mastered, yields productivity rates

which compare well with other "older" languages and contains powerful and easily implemented data structuring. The drawbacks to Pascal as a production language probably can be corrected through the current standardization effort.

#### REFERENCES

1. "RELATABASE Processor System Overview," Naval Ocean Systems Center, San Diego, 1979.
2. "FORMEL 2.0—The FORMEL FORTRAN Preprocessor—User's Manual," Computer Sciences Corp., San Diego, 1978.
3. Ball, M. S., "Pascal-1100," Naval Ocean Systems Center, San Diego, 1979.
4. "Pascal Quality Assurance Standards and Conventions," Computer Sciences Corp., San Diego, 1979.
5. "FORTRAN Quality Assurance Standards and Conventions for the COSR/SES Project," Computer Sciences Corp., San Diego, 1979.
6. "American National Standard Programming Language FORTRAN," ANSI X3.9-1978, American National Standards Institute, New York, 1978.
7. Jensen, K. and Wirth, N., "PASCAL—User Manual and Report," Second Edition (Corrected Printing), Springer-Verlag, New York, 1978.
8. Alagic, S. and Arbib, M. A., *The Design of Well-Structured and Correct Programs*, Springer-Verlag, New York, 1978.
9. "UNIVAC FORTRAN V Programmer Reference," UP-4060 Rev. 2, Sperry Univac, 1973.
10. Borgerson, B. R., Hanson, M. L., and Hartley, P. A., "The Evolution of the Sperry Univac 1100 Series: A History, Analysis, and Projection," *Comm ACM* 21, 1, January 1978 25-43.
11. "Specification for the Computer Programming Language Pascal" (draft), International Organization for Standardization, October 1979.
12. Iverson, K. E., "Operators," *ACM Trans. on Prog. Lang.*, 1, 2 (October 1979), 161-176.
13. "American National Standard Programming Language COBOL," ANSI X3.23-74, American National Standards Institute, New York, 1974.

# UCSD Pascal<sup>™</sup>: a portable software environment for small computers

by MARK OVERGAARD

*SofTech Microsystems, Inc.*  
San Diego, California

## INTRODUCTION

The UCSD Pascal System is a complete program development and execution environment for small computers. Its facilities include text editors and file management utilities, as well as compilers (for Pascal, in particular), assemblers, and a linkage editor. The system is highly portable; versions have been implemented on almost twenty minicomputers and microprocessors. (A concise description of system facilities is provided in an appendix.)

The system was developed under the direction of Professor Kenneth Bowles at the University of California, San Diego (UCSD), starting in late 1974. The author, working first at the University as a graduate student, and more recently with SofTech Microsystems, played a principal technical role throughout the evolution of the software.

The original need was for inexpensive interactive access to a high level language for a large enrollment introductory course in problem solving and computer science. We decided early that we would use small, stand-alone computers as the hardware foundation for our solution rather than larger, time-shared computers. We then chose Pascal as the language to be used by students in the introductory course, and also as the implementation language for system software we would need to build for these small machines. The design goals for Pascal<sup>1</sup> specifically included these two kinds of applications. We used the P-compiler<sup>2</sup> as the starting point for our Pascal implementation.

We needed a stand-alone Pascal program development and execution environment suitable for computer-naive students, but also capable of being used for maintenance of the system itself. We had two primary design concerns:

- 1) a user interface oriented specifically to the novice, but also acceptable to experts;
- 2) a strategy for fitting these facilities in small, stand-alone machines. By our definition, a "small machine" had less than 64k bytes of memory, dual standard floppy disks, and a CRT terminal. We were particularly concerned about the Pascal compiler, since we knew of no implementation of Pascal within those constraints.

These two concerns have been themes of the evolution of

UCSD Pascal ever since. Experts and novices have both continued to use the system, and adaptations of the system to even smaller host configurations (e.g. the Radio Shack TRS-80<sup>™</sup>) have been done.

In the user interface area, our current philosophy is still very similar to that originally developed. Given a single user host computer and a CRT terminal (the preferred environment), our approach is to keep the user continuously informed about the state of the system and the options available in that state. A "prompt-line" is maintained on the terminal screen listing these options. The user can select an option by typing a single-character command. In text editing, the high bandwidth connection to the user is exploited to provide a continuously updated "window" into the text file being perused or modified. A naive user in this environment is led tutorially through an interaction with the system. Experienced users can ignore the continuous status information unless it is needed.

The original small system implementation strategy has also largely survived. Its major component is the use of a p-machine as the foundation of the system. The UCSD p-machine is a simple idealized stack-oriented computer which can be emulated by an interpreter executing in the machine language of a conventional host computer.

The important requirement is that the instruction set of the p-machine be designed so that p-code representations of Pascal programs are very compact and easy for the compiler to produce. Compactness and ease of generation for p-code are both important, in order to minimize the size of the Pascal compiler, which is the biggest single software component of the system.

We were inspired by the code-compaction approach used at Burroughs,<sup>3</sup> but had to adapt it to conventional hardware without facilities for microprogramming or bit-level addressing. Through several iterations on the p-machine architecture, static and dynamic statistics of opcode and operand frequencies were used to identify the instruction sequences occupying the most space and redesign them to be more compact. Tannenbaum<sup>4</sup> has independently pursued the same sort of optimizations, but without our concern for software portability.

More generally, our concern for small host computers is pervasively reflected in our choice of functional facilities



included in UCSD Pascal and the implementation approaches used to provide them.

As we accumulated experience with hardware and software for small systems, it became clear that while the costs of raw hardware would continue inexorably down, the costs of software were an entirely different matter. Thus, a third concern became important in the evolution of UCSD Pascal: conservation of our software investment. The bulk of this paper is devoted to explaining the software conservation strategy that we have evolved.

One final introductory matter needs to be addressed: the unexpectedly large interest in UCSD Pascal from outside the University and how that interest was dealt with. A version of our software running on PDP-11's was first distributed to a few off-campus users in the summer of 1977. Outside interest began to increase when a version for 8080's and Z80's became operational early in 1978. Shortly thereafter, a description of the system in Byte<sup>5</sup> drew over a thousand inquiries. As interest in the software mushroomed through 1978, it became clear that the demand for UCSD Pascal could not be met within the available resources of the University project. For this and other reasons, investigation began at the University into ways in which support of the growing UCSD Pascal user community could be moved off-campus. This effort culminated in June, 1979, with the designation of SofTech Microsystems as the focal point for licensing, support, maintenance and continued evolution of the UCSD Pascal language and system. Advanced development work has continued at the University in the Institute for Information Systems. Some of the results of that work are described later in the paper.

#### CONSERVATION OF SOFTWARE INVESTMENT: AN OVERVIEW

The first component of our effort to conserve software investment is the use of Pascal as the principal system and application language. In 1974, when the choice was made, Pascal was one of the more popular academic languages, and provided the best combination of power and ease of implementation. We certainly did not foresee the current avalanche of industrial interest in the language.

The second component of our software strategy is a heavy emphasis on software portability. We feel that independence of software from differences in the underlying hardware is crucial to small machine applications in order to have:

- 1) the freedom to change hardware to take advantage of rapid technology developments, thus reducing hardware cost or increasing its performance.

Consider the experience at UCSD with equipment for teaching Ken Bowles' introductory computer science course. The first small machines, acquired in the fall of 1975, were nine PDP-11/10's (worth about \$17,000 apiece). Since then, two equipment transitions have occurred at two year intervals: first to 25 Terak 8510a computers (worth just under \$8000 apiece), and

then to 45 Apple II computers (at less than \$3000 each). And all of these configurations can run virtually the same software!

- 2) the chance to reduce the effective cost of software by widespread sharing within an application community.

At this writing there are more than 15,000 computers (with many different host CPU's) running UCSD Pascal. This number is large enough to justify significant investments in application software. High costs to the individual end-user are not required to recoup those investments.

These portability motivations are, of course, not new to the small machine environment; they have just become much more urgent as the cost of software continues to rocket while that of hardware plummets.

There are several different approaches to software portability. One can emphasize the *program*, picking a particular large application, like a data base management package, and working to reduce effort to move it to new host operating system or processor environments. A disadvantage of this approach is that the effort must be expended anew for each additional application considered.

Another possibility is to emphasize a *language* and its implementation. Here the theory is that programs written in that language will port easily (by recompilation) to a new environment after the language itself is moved. Unfortunately, any sizable application program calls on I/O and other operating system resources in ways that may conflict with services available in a new environment. Therefore, changes are likely to be needed in the application programs to be moved. Figure 1 may clarify this approach.

UCSD Pascal provides a portable software *environment*. When the system is moved to a new host, all the conventions about file titles, disk organization, and other operating system matters are replicated. Therefore applications in UCSD Pascal can usually be moved to a new host without any modification to the source version of a program. This is shown in Figure 2. As we will see below, it is even possible to move the object version of a program to a new host.

Software environment portability has been independently pursued with the Thoth<sup>6</sup> and Unix<sup>TM7</sup> operating systems. Neither of these efforts has had our concern for the special problems of small hosts.

In two sections below, specific areas of UCSD Pascal portability are examined. In the first, independence from the host processor (ignoring system peripherals) is considered. Secondly, our approach to independence from host peripheral devices is discussed.

While our concern with portability is long-standing, recognition of the third component of our conservation strategy (organization for support and maintenance) is more recent. As UCSD Pascal entered widespread production use in 1979, it became clear that maintenance and support activities needed careful attention if a large user community using many varieties of host hardware was to be properly served

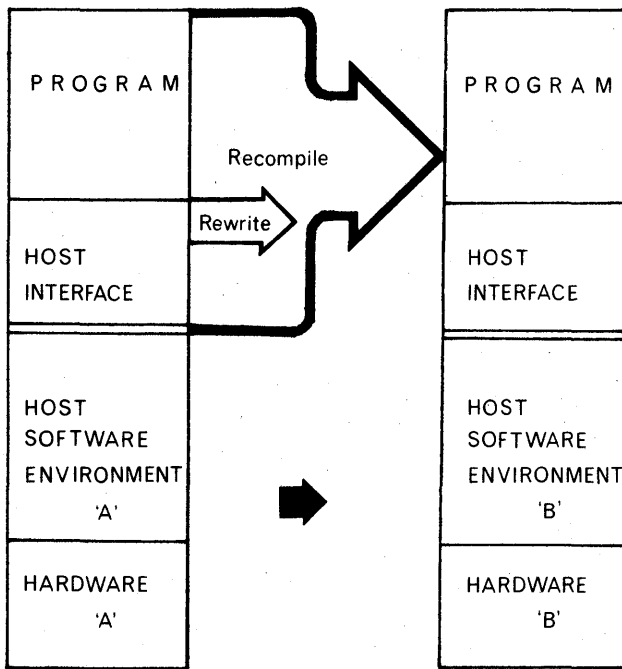


Figure 1—LANGUAGE approach to portability.

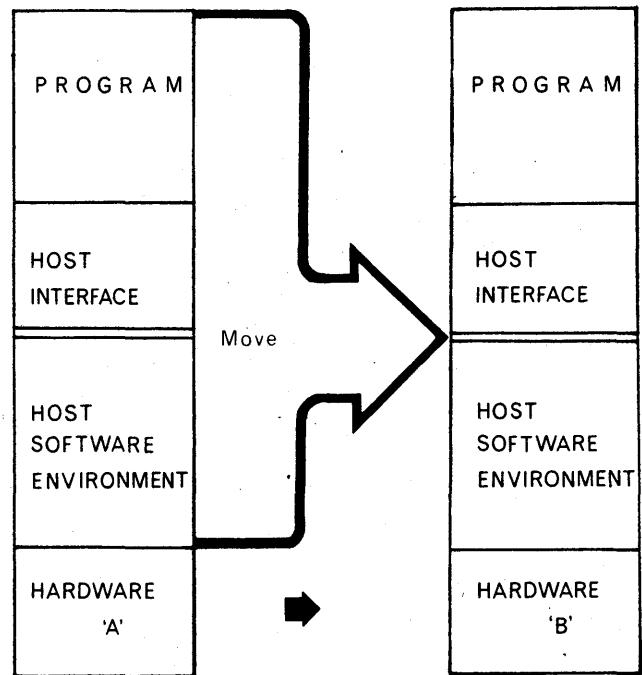


Figure 2—ENVIRONMENT approach to portability.

at a reasonable cost. It also became clear that the academic environment was not well-suited to this task. One reason why SofTech was chosen to take over support of UCSD Pascal was their experience in developing and using software engineering tools of the type needed to support maintenance of UCSD Pascal.<sup>8</sup>

INDEPENDENCE FROM THE HOST PROCESSOR

*P-machine contributions to software portability*

The key to host processor independence is the designation of the p-machine as the foundation of the UCSD Pascal System. One result is that the entire Pascal system, including editors, compilers, operating system, etc., can be moved to a new host computer by reimplementing the p-machine and associated low-level routines in the native language of the new host. The relative simplicity of this implementation task has been widely exploited. Implementations of some variant of the UCSD p-machine have been done for the microprocessors and minicomputers listed below. (Those that have actually become products are marked with an asterisk.)

- \*Data General Nova
- \*Digital Equipment PDP-11<sup>™</sup> & LSI-11<sup>™</sup>
- \*General Automation GA-16
- Hewlett Packard System 45
- \*Intel 8080 & 8086
- Lockheed Sue
- \*Mos Technology 6502
- \*Motorola 6800 and 6809; 6800 is being actively pursued

Nanodata QM-1

- \*Sperry Univac (Minicomputer Operations) V77
- \*Texas Instruments 9900
- \*Western Digital Pascal MicroEngine<sup>™</sup>
- \*Zilog Z80 and Z8000
- several custom micro-coded processors

The effort required for one of these implementations has ranged from 6 person weeks to 9 person months, depending on experience of the implementor and suitability of the host for p-machine implementation. We know of no other body of software as large as the UCSD Pascal System that has been moved to as many different host computers.

UCSD p-code is portable at the *binary* code file level (see more discussion of this under Data Representation Issues). Other pseudo-machine oriented efforts (Janus,<sup>9</sup> for example) have generally standardized, instead, at the level of *symbolic* pseudo-machine assembly language. The choice of a binary interface for UCSD Pascal makes it much more practical for p-code to serve as a sort of "lingua franca," for communicating object programs among a wide user community. Carl Helmers<sup>10</sup> has proposed a distribution approach for small machine application packages in which the last few pages of the user document would contain a printed bar-code encoding of the object program. As he recognized, UCSD p-code fits very nicely into this approach.

P-code is a lingua franca in another sense: even though the p-machine is optimized for Pascal programs, translators can and have been written from FORTRAN and BASIC to p-code. With some strategic additions to the p-machine to support new data types, even a COBOL to p-code translator is feasible.

### *Concentration on small systems*

We have chosen to limit the class of suitable host computers so as to enhance portability within that class. We assume that memory can be viewed as 8-bit bytes and 16-bit words and that the 7-bit ASCII character set is used. Distributed versions of the p-machine are also limited to a 16-bit address space. (A later section discusses removal of this limitation.) Our success in transcending the details of the host environment has been substantially increased by specializing in this limited class. Fortunately, most small computer systems are included.

### *Performance issues*

What price has been paid for these portability benefits? One part of the cost is in the reduced execution performance of interpretively executed p-code compared to other implementation approaches. For many small computer applications (text editing or data capture, for instance) interpretive execution on a dedicated microprocessor is more than adequate. In other applications (e.g. compilation) the benefits due to the small size of p-code outweigh the drawbacks of raw execution speed.

It is also possible (with some reduction in portability) to code time-critical routines directly in assembly language and call them from a high level host program. Most real-time programs can meet performance requirements with only a small portion (less than 10 percent) written in assembly language.

Another performance possibility is to provide more direct hardware support for the p-machine. In the Western Digital MicroEngine, the MOS chip set used in the implementation of Digital Equipment's LSI-11 has been microcoded to implement the p-machine directly. P-code is thus the native language of the MicroEngine. Performance improvement factors of five or more have been measured for the MicroEngine over interpretive execution on the LSI-11. At least another factor of two in performance has been achieved with a micro-coded p-machine based on high speed bit-slice technology. An overall improvement factor of twenty compared to LSI-11 interpretation is probably achievable with standard low-cost components. (All of these comparisons apply to simple integer arithmetic and array operations, as in an integer sort routine.)

### *Data representation issues*

Some concessions to efficiency over portability have been made. The biggest has to do with representations of p-machine data types. Although a standard on floating point number representation and algorithms is being developed by the IEEE,<sup>11</sup> there are still many different formats supported by various vendors. We have standardized on a size of 32 bits for floating point numbers, but do not require a particular representation. Therefore, advantage can be taken of existing floating point software and hardware support. Where

new floating point implementations have been done (on the 6502, 6800, 9900, and W.D. MicroEngine) the IEEE format is used. A machine-dependent "power of ten" table allows all high level software (even conversions between ASCII and internal floating point) to be isolated from knowledge of the internal representation. Long integer representations can also vary. Binary integer, packed BCD, and radix 100 are among the feasible representations. A dramatic performance improvement can be achieved on some hosts by choosing a suitable representation. Once again, higher level software in the Pascal system need only care about the *length* (in words) of a long integer.

Even ordinary 16-bit integers do not have a standard representation (at least in the ordering of their two bytes). In some host architectures (e.g. PDP-11, 8080) byte zero of a word is the least significant byte; in others (e.g. 6800, Z8000, IBM 370), byte zero contains the most significant bits of the 16-bit integer. Thus, the interpretation of two adjacent bytes as a 16-bit word is machine-dependent. Here, too, an attempt to force a single representation on all architectures would be prohibitively expensive.

What is the impact on portability of these representation decisions?

First, portability of source programs is not affected, unless a program specifically chooses to deal with the representation, by bypassing the type philosophy of Pascal. We need, but do not yet have, an analog of the LINT program under Unix,<sup>7</sup> which could comb source programs for potential trouble spots of this type.

Second, data files containing any of these data types are not directly readable by implementations with different representation choices. It is possible to design file record structures so that an application program can automatically compensate for representation differences.

Third, code files are sensitive to the byte ordering (we call it "byte sex") of the host processor. The dependence occurs where adjacent bytes in the code stream represent words (mostly in superstructure tables). The Pascal compiler can generate code files of either type on any host, and a utility program is provided to convert object files from one type to another.

Finally, code files containing floating point constants are not directly movable between hosts with different floating point representations. It is usually possible, by doing some computation at run-time, to avoid this problem.

## INDEPENDENCE FROM THE HOST PERIPHERALS

### *UCSD Pascal I/O hierarchy*

Our approach to achieving peripheral independence is to provide a hierarchy of I/O environments. The levels of the hierarchy are chosen to further two objectives:

- 1) isolation of application programs and most system components from details of the host computer, and
- 2) reduction of effort involved in adapting to new host configurations.

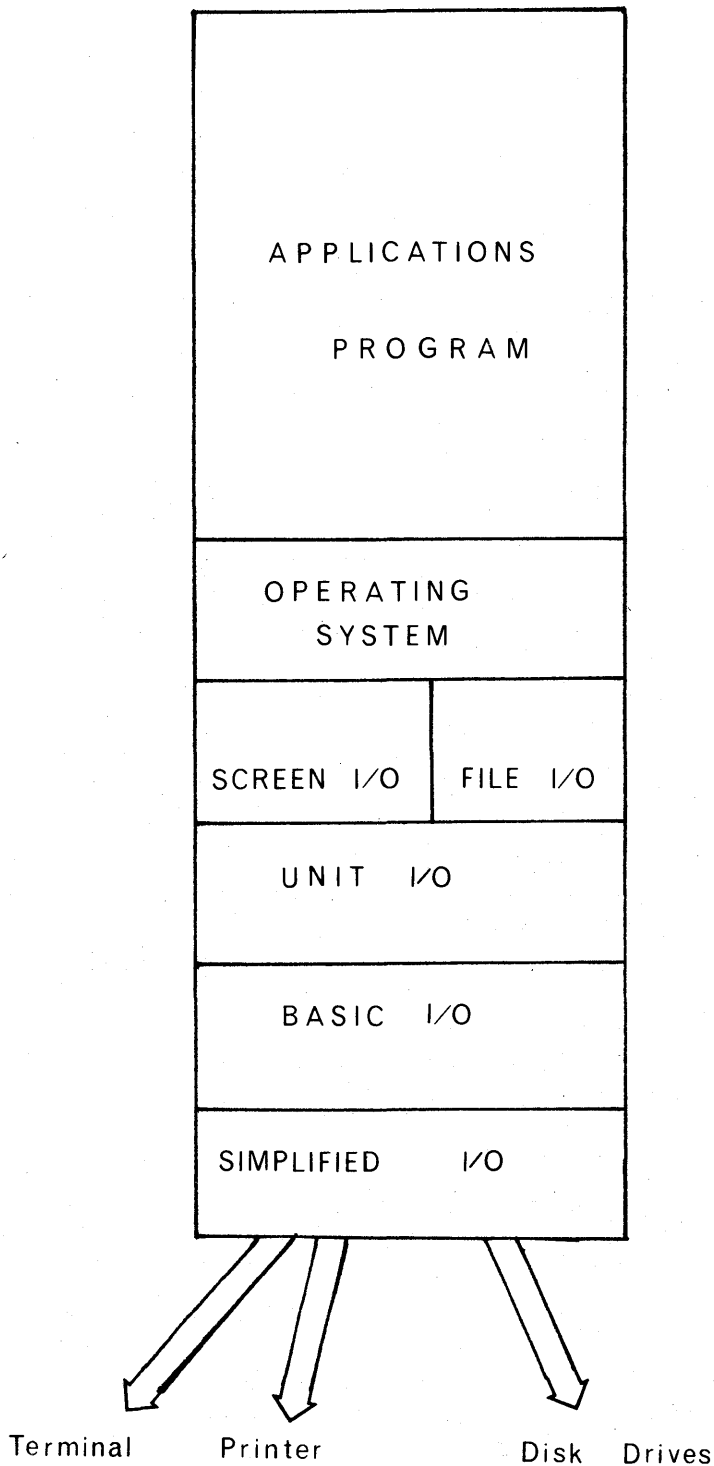


Figure 3—UCSD Pascal I/O hierarchy.

The levels we identify are pictured in Figure 3 and listed and described below:

- 1) Screen I/O. This level presents a uniform image of a screen terminal. Capabilities include moving the terminal cursor; clearing all or part of individual lines or

- the entire screen; and accepting cursor control or other special commands from the terminal keyboard.
- 2) File I/O. At this level, devices are designated by logical volume names. Volumes can be serial (e.g. console) or random access (e.g. disk). Random access volumes can have directories of named files. Serial volumes are (possibly bi-directional) byte streams. Textual I/O (with full conversion between internal representations and ASCII) is provided from and to volumes or files. Record-oriented I/O (with automatic blocking and deblocking) is provided with random access volumes or files.
- 3) Unit I/O. At this level, devices have numbers that indicate their type (e.g. 1 for console terminal, 6 for printer, etc.). A serial device is still a byte stream, with knowledge of a few special output characters (blank compression codes, carriage returns). A random access device is considered an array of directly addressable 512-byte blocks. No knowledge of files, textual or record-oriented I/O is available at this level.
- 4) Basic I/O. Capabilities of random access devices at this level are similar to those available at the Unit I/O level. Serial devices are much simpler. Serial transfer occurs one character at a time and no special output characters are processed. Special input characters from the console terminal can cause console output to be stopped, restarted or discarded.
- 5) Simplified I/O. Here the random access interface is much more primitive. A device is viewed as sequence of tracks, each containing an array of physical sectors. Transfers occur, one sector at a time, between a main memory buffer and a (track, sector) coordinate. Serial device capabilities are similar to those in Basic I/O, except that no special input characters are recognized.

Table I summarizes the degree of independence from host configuration provided at each of these levels. The Basic and Simplified I/O components are generally implemented in assembly language and clearly dependent on peripheral details. The pseudo-machine interpreter (which generally includes Unit I/O) is dependent on the host processor, but defers peripheral details to lower levels. File and Screen I/O are implemented in Pascal and are therefore independent of both peripheral and processor variations. The application program does not have to depend on anything but the virtual environment provided by the Pascal system.

TABLE I.—Host configuration independence

Software level	Is independent of:		
	Console Terminal	Central Processor	Peripheral Complement
Application program and most system components	Yes	Yes	Yes
Screen I/O	No	Yes	Yes
File I/O	Yes	Yes	Yes
Interpreter and Unit I/O	Yes	No	Yes
Basic and Simplified I/O	Yes	No	No

### Adaptation to host configurations

Four kinds of user adaptation are intended. The easiest adaptation is also by far the most frequently needed; catering to console terminal peculiarities. Most of the effort involved goes to changing the entries in terminal description tables maintained by the system (a program, SETUP, is provided to do this). Some programming (in Pascal) of interface procedures may be required. Average effort involved for the user is an hour or two. A test program is provided to determine if the effort was successful.

Next in order of increasing difficulty and decreasing frequency of need is adaptation of the Simplified Basic I/O Subsystem (SBIOS). Given that a user is knowledgeable about his peripheral interface, and has access to existing low-level driver software, a few days of work should be sufficient to bridge the gap between those drivers and the SBIOS interface, producing a usable Pascal System. Again, a test program is provided. The approach we have taken here is based on that developed by Digital Research for the CP/M<sup>®</sup> operating system.

The Basic I/O Subsystem (BIOS) is comparable to its simplified cousin, except that it has more responsibility, and therefore more possibility for optimization. For instance, it is possible in the BIOS to take full advantage of a direct memory access interface to disk. The BIOS definition emphasizes performance and flexibility, while the emphasis with SBIOS is on ease of adaptation. BIOS implementation generally takes a week or two of effort, assuming detailed knowledge of the peripheral complement and familiarity with our I/O structures.

Finally, the most elaborate adaptation is to a new host processor (which is included in this peripheral independence section for completeness). As mentioned above, effort involved here is more than a month, but less than a year; six person months is probably a good average.

Table II gives the various levels of adaptation. For each, a reasonably realistic level of effort is given, as well as an approximate "probability of need" indicating how frequently adaptation at that level will be desired. Fortunately for the viability of our approach, these probabilities have some basis in reality.

## ENHANCEMENTS

### Native code generation

Active work is under way at UCSD to provide the ability to translate selected procedures of a p-code program to native code for a conventional host computer. This possibility will alleviate many of the performance drawbacks of our p-code orientation without sacrificing portability. Programs can be written and maintained entirely in Pascal, and the p-code object version is still transferable among different kinds of host computers. If active use of a program reveals performance bottlenecks, the time-critical procedures can be translated to a native code.

TABLE II.—Intended levels of adaptation

Level of adaptation	Approximate Effort Involved	"Probability of need"
Application program	very little, if any	1.0000
Screen I/O	hours	0.1000
Simplified I/O	days	0.0100
Basic I/O	weeks	0.0010
Interpreter	months	0.0001

Code generation is implemented as an optional step in the compilation process. It takes as input a complete p-code program and produces, as output, a mixture of unmodified p-code and translated native code procedures. This process is diagrammed in Figure 4. Internally, the code generator represents a p-code procedure as a forest of expression trees. Several traversals of these trees occur during the translation process.

Note that the p-code input to the code generator can come from a Fortran or Basic compiler, as well as from the Pascal compiler. Once again, software investment is conserved, since only one code generator for a particular target machine serves several languages.

Implementations are in progress of code generators for the PDP-11, TI9900, Intel 8080, Mos Technology 6502, and General Automation GA-16. At this writing, the first two versions are farthest along. For both processors, translated native code is about 50 percent larger than the corresponding p-code. Improvements in execution performance compared to interpretive execution on the same host have been around a factor of 10 for the PDP-11 and a factor of 15 for the TI9900.

### New definition of "small" systems

As new microprocessor architectures (e.g. 8086, Z8000, 68000) and peripherals (e.g. low-cost Winchester disks) become widely available over the next several years, the definition of "small" low-cost computer systems will have to broaden considerably to include those with megabytes of main memory and tens of megabytes of mass storage. We are investigating ways in which the UCSD Pascal system could allow these facilities to be exploited. The situation is complicated by our need to continue supporting *both* Z80 class and Z8000 class users for the foreseeable future. Therefore, it must be possible for user programs, and particularly system components, to run in both environments. It must also be possible, of course, to produce a user program that requires so many resources that it can only be run in an expanded environment.

As an example of the extension approach we are pursuing, consider the problem of dealing with address spaces bigger than 16 bits. In the near term, it is quite easy to apply the 16-bit limitation only to *data* space: object code need not be accessible within the 16-bit area. For the longer term, we are considering a scheme in which details of physical ad-

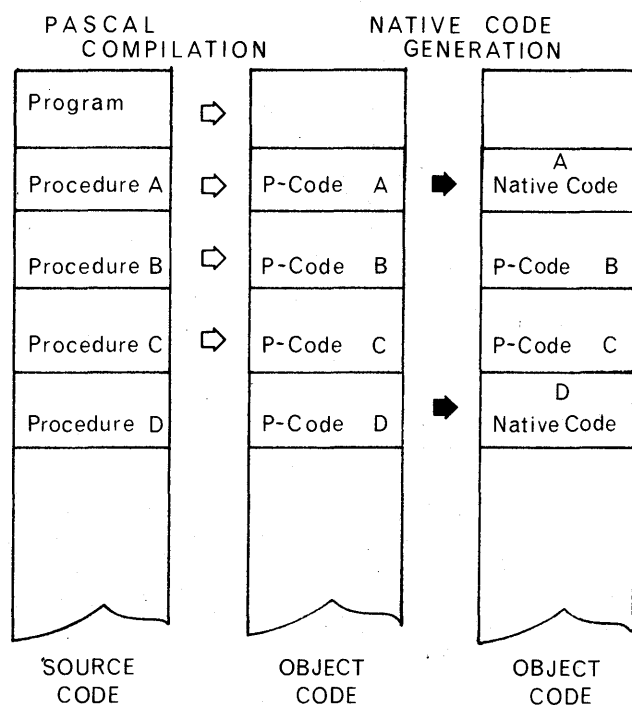


Figure 4—Selective native code generation.

addresses in the host are not important to higher level software. That software is already isolated from whether host memory is addressed as words or bytes. We propose further isolation: only a small machine-dependant portion of an implementation would know whether addresses are 16 or 32 bits in size and the detailed meanings of those bits.

## CONCLUSION

The UCSD Pascal language and system have demonstrated that sophisticated, yet accessible, high level language capabilities can be provided in constrained microcomputer environments. More importantly, we have been shown that these facilities can be compatibly implemented on a wide range of host computers. As the inexorable advance of semiconductor technology continues, we believe that conservation of software investment is the fundamental practical problem facing the microcomputer industry. We think our efforts are contributing to the solution of that problem.

## ACKNOWLEDGMENTS

The UCSD Pascal system is the work of a large group, too numerous to list here, of graduate and undergraduate students at UCSD. The role of Kenneth Bowles in inspiring, directing, and energizing this work has certainly been crucial to its success. Helpful comments on this paper from John Brackett, Winsor Brown, Al Irvine, and Richard Kaufmann are gratefully acknowledged, as is Keith Shillington's preparation of the illustrations.

## REFERENCES

1. Wirth, N., "The Programming Language Pascal and its Design Criteria," InfoTech State of the Art Report - High Level Languages, 1972.
2. Nori, K. V., U. Ammann, et al., "The Pascal (P) Compiler: Implementation Notes. *ETH Zurich Technical Report 10*, December 1974.
3. Wilner, W. T., "Design of the B1700," *Proceedings of AFIPS*, Fall 1972.
4. Tannenbaum, A. S., "Implications of Structured Programming for Machine Architecture," *Communications of the ACM*, March 1978.
5. Bowles, K. B., "A (Nearly) Machine Independent Software System for Micro and Mini Computers," *Byte*, May 1978.
6. Cheriton, D. R. and M. A. Malcolm, et al., "Thoth, a Portable Real-Time Operating System," *Communications of the ACM*, February 1979.
7. Johnson, S. C. and D. M. Ritchie, "Portability of C Programs and the Unix System," *Bell System Technical Journal*, July-August 1978.
8. Eanes, R. S. and C. K. Hitchon, et al., "An Environment for Producing Well-Engineered Microcomputer Software," Fourth International Conference on Software Engineering, 1979.
9. Waite, W. M. and B. K. Haddon, "A Preliminary Definition of Janus," *University of Colorado Technical Report SEG-75-1*, 1975.
10. Helmers, C., "Editorial," *Byte*, August 1978.
11. Gustavson, D. B., "Standards Committee Activities: An Update," *Computer*, July 1979.
12. Jensen, K. and N. Wirth, "Pascal User Manual and Report," New York: Springer Verlag, 1974.
13. Ravenel, B., "Toward a Pascal Standard," *Computer*, April 1979.

## APPENDIX: Facilities of the System

### *Program execution environment*

The foundation of the System is the UCSD p-machine. It is a simple idealized stack computer which can be implemented either by direct hardware support (as in the Western Digital MicroEngine) or by an interpreter executing in the machine language of a conventional host computer. On a conventional host, a single object program can include both p-code (to be interpretively executed), and native code (for direct execution by the host).

Peripherals are accessed by logical "volume" names. Serial volumes (e.g. console terminal) are considered byte streams. Random access volumes (e.g. floppy disk) can have directories of named files. Various kinds of logical transfers involving volumes and files are supported.

### *Program execution and file manipulation commands*

The user can execute a named object program, or use short cut commands to invoke important system programs. The user can also designate individual files or groups of files for removal, renaming or transfer among on-line devices. Other commands support various housekeeping needs: listing directories, compacting the files on a disk, and testing disks for invalid areas. Finally, the user can designate a "work file." Subsequent editing, compilation and execution commands apply to this work file by default.

### *Text preparation and modification facilities*

Two styles of text editing are supported: one requires a video display terminal, and the other does not.

When the system console is a CRT, the "screen-oriented" editor can usually be used. This editor maintains a cursor into the text file being edited and a "window" into that area of the file on the terminal screen. Modifications to the text are made by the intuitive and mechanical process of moving the cursor to the site where change is desired and indicating the change. Commands are provided for moving the cursor, finding and replacing textual patterns, making insertions and deletions, and copying text into the cursor position from elsewhere. Special facilities exist for processing documents. User-specified left and right margins can be automatically enforced by the editor and new margin requirements can easily be applied to existing text.

The second available style of editing does not require a screen terminal. Once again, a cursor is maintained, where most of the action occurs. But the user is responsible for maintaining a mental image of the cursor context. Commands are available for insertion, deletion, and copying of text, as well as for moving the cursor. A simple macro facility is provided.

### *Programming languages*

The principal programming language supported is UCSD Pascal. Except for the provision of procedures as parameters, UCSD Pascal is largely consistent with the base Pascal language, as defined in Jensen and Wirth's User Manual and Report.<sup>12</sup> UCSD Pascal is also quite consistent with the emerging international standard for Pascal.<sup>13</sup> We are committed to eventual complete compliance with an adopted standard.

UCSD Pascal includes various extensions beyond the base language. We summarize, here, the most important:

- 1) Dynamic character strings. A predeclared type "string" is supported. A string variable contains a sequence of characters. A maximum length for the sequence is specified in the declaration. Concatenation of strings; insertion, deletion and extraction of substrings; and string pattern matching are provided by predeclared service routines.
- 2) Encapsulation and separate compilation. A new composite declaration, the "unit," is provided. A unit is a group of procedures, functions, and data structures, usually related to a common task area. A program or another unit (a "client module") can access these facilities by naming the unit in a simple "uses" declaration. A unit consists of two parts, the interface part, which can declare constants, types, variables, procedures and functions that are public (made available to any client module), and the implementation part, in which private declarations can be made. These private declarations are available only within the unit, and not to client modules. Units can be compiled separately from their client modules.
- 3) Extended precision integers. A "long integer" data type is provided. Integers up to 36 decimal digits in size can be represented and participate in the standard in-

teger operations: addition, subtraction, multiplication, and division. Conversions among long integer, string and standard integer forms are provided.

- 4) Concurrent processes. Another type of routine in UCSD Pascal is the "process." Processes are declared with the global procedures of a program and have the same lexical access to global variables and procedures. A process is different from a procedure in that when invoked, it proceeds in parallel with its invoker. Semaphore variables, plus wait and signal primitives, are provided to allow these parallel processes to synchronize and communicate reliably. With the "attach" procedure, a semaphore can be associated with an external interrupt. This association causes the semaphore to be signaled if the interrupt is activated. Thus Pascal processes can respond to external events.
- 5) Miscellaneous extensions. Other additions to UCSD Pascal provide random access to Pascal file components and a constrained interprocedural goto mechanism. Segment routine declarations allow designation of overlays and external procedure declarations allow an assembly language routine to be called from a Pascal host as if it were a Pascal procedure.

A Basic compiler exists for the UCSD Pascal system, but is not currently being supported.

Assembly language is available for most processors on which the system is supported. The assemblers can be used for stand-alone programs (such as interpreters) or for procedures which will be bound into high level language host programs. The approach is to provide (as far as possible) the syntax for machine instructions defined by the original processor manufacturer (e.g. Zilog for the Z80). A common syntax has been defined for assembler directives and assembly-time expressions. Naturally, all of the assemblers can run on any host processor variant of the System, so a single type of host can be used to support assembly language programs for multiple machines.

Directives supported include the usual facilities for macro definition, conditional assembly, storage allocation and listing control. Additional directives allow communication with external labels in other assembly language routines. Finally, special provision is made for communication between an assembly language routine and a Pascal host program. The low-level routine can request access to host program global variables and constants. It can also allocate its own global storage space.

### TRADEMARKS

UCSD Pascal is a trademark of the Regents of the University of California. PDP-11 and LSI-11 are trademarks of Digital Equipment Corporation. Unix is a trademark of Bell Laboratories. MicroEngine is a trademark of Western Digital Corporation. CP/M is a trademark of Digital Research, Inc. Radio Shack and TRS-80 are trademarks of Tandy Corporation.

## Software Quality

The FAA's computerized Enroute System for controlling in-flight commercial aircraft crashes during a peak holiday period due to overloading. The DoD Early Warning System, a computerized air defense system, mistakes the rising moon for a barrage of incoming enemy missiles and shock waves travel all the way to the White House. A single erroneous statement in a small computer on-board a French weather satellite causes 71 of 142 weather balloons to self-destruct. These experiences would not have happened if there had been better software quality.

Assuring software quality has been and still is a thorn in the side of most software customers and project managers. This phenomena crosses all customer boundaries: commercial, industrial, military, other government; and crosses different application types: operating systems, information systems, process control, command and control, communication, business systems, etc. The "Software Quality" area contains four sessions that will enlighten both purchasers and developers of software with discussions and papers which will reveal not only current quality-related problems, but also suggested solutions.

Dr. Edward Miller will chair a panel session on Software Quality Testing. Panelists will discuss the need for establishing and following quality standards, and programming and testing techniques to improve the testing process.

Dr. Ned Chapin will chair a three-paper session on Software Quality Metrics. "Measuring program complexity in a COBOL environment" by Zolnowski and Simmons presents a composite measure of program complexity that provides an objective quantitative evaluation for any program or programming effort. Another paper, "The complexity of an individual program" by John McTap critiques the Zolnowski and Simmons model and proposes a model extension. The third paper, "An information theory based complexity measure" by Eli Berlinger proposes a measure of programming difficulty based on the probability with which various tokens of a program are used.

Dr. Leonard Gardner will lead a panel discussion of accomplishments, problems, and proposed solutions of present and future software standards. These will encompass machine, assembly, and high level languages, and software related to buses and their interfaces. Panelists have been selected who represent a very broad base of standardizing activities of various technical societies and workshops.

Mr. Kurt Fischer will lead a panel discussion of current software quality assurance problems and techniques. Topics of discussion will include the purpose of software QA, the techniques that are currently used, the benefits that are received from QA programs, and future directions that software QA should take. The selected panelists have all managed QA programs on major projects and will be glad to share their experience with the audience.



Kurt F. Fischer  
*Area Director*





# Measuring program complexity in a COBOL environment

by JEAN ZOLNOWSKI and DICK B. SIMMONS

*Texas A&M University*  
College Station, Texas

## INTRODUCTION

Webster<sup>1</sup> defines complexity as the "quality or state of being: hard to separate, analyze, or solve; complicated; intricate; involved; having confusingly interrelated parts." This definition is certainly apropos of software and in fact the term program complexity is found often in discussions centered on software and its evaluation. An understanding of computer program complexity is considered necessary in comprehending how software is written and how a language is used.

In particular, a figure of merit in the form of a complexity measure for a program can serve as a factor in evaluating the efficacy of various programming styles/methods; in producing programmer productivity measurements; in producing software cost estimations based on a programmer's past program complexity history; in relating errors to specific program characteristics; and in similar types of quantitative analyses.

This paper describes a complexity measure, a composite index of complexity, which insures that the relative merits of a program will be judged/compared not according to untested hypotheses or programmers' subjective judgments but rather according to complexity factors actually evidenced in the program and therefore relevant in measuring it. For example, this index of complexity has been used to compare structured versus unstructured programming styles within a reference group of COBOL programs. The measure verified that in fact the structured programming group did produce less complex programs.

The index is a complexity measure based upon the assumption that opinions of experienced authors reflect relevant aspects of program complexity. The method for relating the diverse set of authors' proposed complexity characteristics in a measure is based on Gunning's<sup>2</sup> idea of a fog index, i.e., a score is assessed against a program based on complexity characteristics evidenced in the program and specific weighting factors attached to each complexity characteristic. The index of complexity evaluates a program within a reference group of programs reflective of typical program types and programming styles of the particular programming environment in which the measure is being used.

The sections to follow will outline how the procedure for producing this index of complexity can be accomplished in

an arbitrary programming environment. An application to a reference group of 13 COBOL programs, typical of medium-sized production programs in a computing center environment, will be presented. Also, discussions will be oriented toward the actual use of the index and the data from which it is derived in evaluating structured versus unstructured COBOL programming styles.

## CURRENT APPROACHES TO PROGRAM COMPLEXITY

The difficulty in understanding complexity is apparent in the diversity of opinions that exist concerning the causes of a program's complexity. Discussions on program complexity are centered in essentially three areas in the literature: cost estimation techniques, authors' discussions on the types of program characteristics each considers relevant to complexity, and specific complexity measures which have been proposed. Table I lists some proposed measures (References 3 through 17).

Several of these complexity measures concentrate on specific structural characteristics of a program. This can present several problems in producing a relevant figure of merit for a program.

One obvious problem with the use of a measure defined in terms of a single specific structural characteristic or a generalized categorization is that, although it may reflect some minimal differentiation between programs, it may not give the most accurate or sufficient picture of the particular aspect of the program it purports to measure. This can be illustrated by considering three programs from the COBOL reference group. Each program contains approximately the same number of verbs but each was written in a different structured programming style (definitions of each style are contained in Table II).

Table II contains data on some specific structural characteristics evidenced in each of these three programs. As this table illustrates, an emphasis strictly on IFs and number of conditions does not provide the best profile of the unique structural aspects of each program.

For the controlled structured program, structure is best evidenced in a detailed analysis of IF nesting. However, the structure of the uncontrolled structured program is best seen through a detailed analysis of its use of the PERFORM verb.

TABLE I.—Proposed Complexity Measures

Orientation of Measure	Description	Author
Control Flow	Cyclomatic Number	McCabe [3]
	Count of Program Paths	Sullivan [4]
	Enhancement of Cyclomatic Number (includes a count of logical conditions)	Myers [5]
	Measurement by the Pair (Cyclomatic Number, Operator Count)	Hansen [6]
	Number of Multiple Entry Loops	Peterson [7]
	Number of Knots	Woodward, et al [8]
	Cyclomatic Complexity Interval Plus # lines into/out of line of code	Cobb [9]
Module Interaction	Number of Modules or Subsystems	Gilb [10]
	$R \left( \frac{\text{Number of Module linkages}}{\text{Number of Modules}} \right)$	Gilb [10]
	Measured based on control structures and control variables	McClure [11]
Data Reference	Measure of difficulty in understanding software's function based on components of sets of input and output	Chapin [12]
Program Control	Minimal Intersection Number	Chen [13]
Logical Complexity	$R \left( \frac{\text{Number of non-normal exits from a decision statement}}{\text{Total number of instructions}} \right)$	Gilb [10]
Software Science	Metrics of software science predict complexity of a program	Halstead [14]
	Approach complexity via statistical (natural) language theory	Shooman & Laemmel [15]
Composite Measure of Complexity	Index of Complexity based on Structure/Interaction/Instruction Mix/Data Reference Program Characteristics	Zolnowski & Simmons [16]
	Interface complexity/Computational complexity/I/O complexity/Readability	Thayer [17]

The unstructured program requires an analysis of GO TO induced loops.

#### METHODOLOGY FOR AN INDEX OF COMPLEXITY

The index of complexity listed in Table I is a measure which reflects the relevant aspects of a program's complexity. The index is based upon the assumption that opinions of experienced authors reflect what is essential to program complexity and therefore form a reasonable basis for a measure of complexity.

The technique is to measure a program using complexity variables, derived from the basis set of authors' opinions, which are relevant to the program in its specific environment and language. The index of complexity for a program is computed within a set of programs chosen as a reference group. Objective data collection and program analysis are used to validate the basis set of complexity variables by showing

which of these proposed complexity characteristics appear often enough in the reference group of programs to be relevant in a measure. For as Knuth's<sup>18</sup> study showed, often what is thought about programs is not actually what appears in programs.

The method for relating the set of complexity characteristics together in the index of complexity is to assess a score against a program based on: (1) a set of complexity characteristics seen to be discriminating within the reference group of programs and; (2) weights assigned to these discriminating complexity variables.

The following series of steps provide an example of how the index of complexity can be computed for a sample group of COBOL programs: Steps 1 thru 5 are concerned with choosing figures of merit (reference values) against which individual programs can be evaluated. Steps 6 and 7 are concerned with actually producing a unique index of complexity for a program.

*Step I. Select a set of program characteristics (features) to use in a measure*

Before actual measurement, what to measure must be known. Unfortunately, there is little empirical data to substantiate any one particular opinion on which program characteristics constitute a definition of program complexity. Therefore, an analysis was done over a wide range of authors' opinions.

Some of the program characteristics said to be relevant

to complexity were easily translated to a variable measurable by a static or manual analysis of a COBOL program. Other opinions on complexity were not so easily measurable. Authors often tended to be rather obscure as to what specific program characteristics reflected their opinions.

Program complexity characteristics derived from authors' opinions were classified under four categories of program complexity: instruction mix, data reference, interaction/interconnection, and structure/control flow. Table III contains examples of the types of COBOL program characteristics listed by category.

TABLE II.—Differences in Programming Styles Between Three Programs within the COBOL Reference Group

Program Characteristic	Controlled* Structured	Uncontrolled** Structured	Unstructured***
# verbs	334	324	336
# IF ... ELSE	50	35	7
# IF (no ELSE)	2	33	47
Avg. # Logical Conditions	1.9	1.9	2.4
# independent IFs	16	43	51
Avg. # instructions w/in IF	12.6	4.8	1.5
% GO w/in IF	0	5.3	52.6
% PERFORM w/in IF	8.9	21.2	1.3
# total PERFORMs	20	24	1
# PERFORMed independently	6	2	1
MAX nesting BREADTH of PERFORM	2	35	0
Avg. # PERFORMs w/in Nested PERFORM	4	110	0
# GO TO induced loops	0	0	26
Avg. # paths in loops	-	-	644.4

\*programmers were forced to utilize the concepts of structured programming. Structured walkthrus, etc. enforced the use of these concepts.

\*\*programmers utilized structured programming subject to their own interpretation of how to implement these concepts.

\*\*\*no attempt was made to adhere to any of the structured programming concepts.

TABLE III.—Complexity Variables

Category	General Types of Variables Measured
1. Instruction Mix	<ul style="list-style-type: none"> <li>- program size data</li> <li>- numbers and types of instructions</li> <li>- specific attributes of instructions</li> <li>- detailed profile of IF nesting</li> </ul>
2. Data Reference	<ul style="list-style-type: none"> <li>- numbers and types of variables</li> <li>- numbers of references to and span of each variable</li> <li>- parameter nesting data</li> </ul>
3. Interaction/Interconnection	<ul style="list-style-type: none"> <li>- numbers of subprograms referenced</li> <li>- number of entry points</li> <li>- types and numbers of parameters</li> <li>- how data connected between modules</li> </ul>
4. Structure/Control Flow	<ul style="list-style-type: none"> <li>- detailed PERFORM flow analysis</li> <li>- counts/attributes of branching instructions</li> <li>- flow graph of the program</li> <li>- basic flow graph variables such as numbers and sizes of basic blocks and intervals</li> <li>- detailed loop analysis</li> <li>- detailed strongly connected region analysis</li> </ul>

#### *Step II. Select a reference group of programs*

The index of complexity proposes a measure for a program within a group of programs. Therefore, the set of sample programs chosen as the reference group has to be reflective of the programming environment in which the software is produced. The set of reference values against which individual programs will be evaluated is predicated on this group.

For example, this research was conducted in a university environment. The learning experiences evidenced in students' programs were not desired. Therefore, sample programs were requested from full-time programmers at the Computer Center and only production programs currently in use were chosen.

Five programmers from different COBOL applications' areas provided programs. Each programmer was asked to give a complexity rating to his/her program, as well as other

details about the environment in which the program functioned. This provided an initial assessment of each program. Table IV illustrates the types of COBOL programs selected as a reference group. Note that Column 6 indicates the programming style in which each program was written.

#### *Step III. Analyze/collect data within the reference group*

A static analyzer, written in SNOBOL, was used to show which proposed complexity characteristics outlined in Table III appeared often enough in the reference group of programs to be relevant in a measure.

There were a large number of features analyzed for each program. The differences in the types of complexity characteristics collected (means, medians, percentages, simple counts, etc.) produced widely varying ranges of values for the different types of variables. Size is obviously a discrim-

inator between programs and this was reflected in the range of values for program size characteristics, such as counts of variables used. Program characteristics such as counts of loops in programs, span of data reference, parameter nesting, etc., were not necessarily reflective of size.

*Step IV. Reduce to a set of discriminating characteristics*

The next step in the process is to reduce the number of features to be used in the index of complexity. The results of the static analysis of Step III provide a profile of each program (a feature vector) consisting of data from each of the four proposed complexity categories.

Certain of the variables, such as sizes of basic blocks<sup>19</sup> and numbers of branches from basic blocks, produced a minimal discrimination between the COBOL programs, even

though these variables are considered relevant to complexity. Other variables, such as those dealing with GO TO statements and looping phenomenon, did prove relevant to the programs' complexity as expected.

Within each of the four complexity categories, a total of 44 characteristics were seen to be discriminating complexity variables for the COBOL reference group under analysis. Variables chosen as discriminators were essentially in two categories: those where programmer insistence or a multitude of authors' opinions necessitated their inclusion and those that obviously discriminated within the sample set. For example, programmer P<sub>5</sub> gave Program 12 a relatively high rating despite its small size. This was due to the programmer's emphasis on the complexity of his input/output manipulations exemplified by the use of the Declaratives Section. Therefore, a variable such as the number of USE

TABLE IV.—COBOL Reference Group Questionnaire Data

Programmer #	Program #	Complexity rating by programmer	Rank according to size	Program type	Special techniques used	Part of larger system
P <sub>1</sub>	1	1	1	Stack maintenance	Structured, * modular, top-down design	yes
P <sub>2</sub>	2	2	7	Data entry and update	Structured,**modular	yes
P <sub>3</sub>	3	4	6	Data base maintenance	None***	yes
P <sub>1</sub>	4	5	4	Parser and subcommand processor	Structured, * modular, top-down design	yes
P <sub>4</sub>	5	5	5	Character manipulation	Structured, * modular, top-down design	yes
P <sub>4</sub>	6	5	3	Data base management	Structured, * modular, top-down design	yes
P <sub>3</sub>	7	5	9	Data base loading	None ***	yes
P <sub>1</sub>	8	7	10	Parser/driver	Structured, * modular top-down design	yes
P <sub>2</sub>	9	7	11	File update	Structured, ** modular	yes
P <sub>4</sub>	10	7	8	Data base management	Structured,* modular, top-down design	yes
P <sub>3</sub>	11	7	12	File-editing, report producer	None ***	yes
P <sub>5</sub>	12	8	2	I/O	None	yes
P <sub>2</sub>	13	9	13	Data collection and accounting	Structured,** modular	yes

\*indicates programmed in a controlled, structured environment. (See Table 2 for definitions)

\*\*indicates programmed in an uncontrolled, structured environment. (See Table 2 for definitions)

\*\*\*indicates programmed in an uncontrolled, environment. (See Table 2 for definitions)

TABLE V.—Instruction Mix Complexity Rating for COBOL Reference Group

Program number	1	2	3	4	5	6	7	8	9	10	11	12	13	Average
Programmer	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>1</sub>	P <sub>4</sub>	P <sub>4</sub>	P <sub>3</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>4</sub>	P <sub>3</sub>	P <sub>5</sub>	P <sub>2</sub>	
# USE	0	0	0	0	0	0	0	0	0	0	0	1	0	.1
# SORT	0	0	0	0	0	0	1	0	0	0	4	0	0	.4
# SEARCH	0	0	0	0	0	1	0	0	0	0	1	2	0	.3
% COMMENTS	0	0	0	9	6	7	0	5	0	23	0	0	0	3.8
% PERFORM	0	33	9	11	13	9	.2	14	15	6	11	6	17	11.1
% IF	8	21	20	20	18	13	16	14	23	16	19	13	18	16.9
% GO	0	3	11	1	1	0	20	4	9	0	13	16	4	6.3
Maximum depth of IF nest	0	3	2	3	5	5	2	4	5	10	2	3	5	3.8
# sections	0	6	0	0	0	0	2	0	9	0	7	0	9	2.5
Size (verbs)	12	324	213	141	180	120	336	347	529	334	564	104	836	310.8
Avg segment size statements	8	16	12	10	14	19	7	11	8	23	6	4	20	12.2
Avg segment size - verbs	6	14	10	7	9	8	7	7	7	10	7	4	18	8.8
Total score	1	7	5	2	5	3	4	3	7	4	8	4	8	
Relative complexity rank for instruction-mix	1	10	8	2	8	3	5	3	10	5	12	5	12	
Unique complexity rank	1	10	8	2	9	3	5	4	11	6	12	7	13	

verbs was included as a discriminator in the instruction mix category.

Tables V thru VIII provide a sample of the complexity characteristics found to be discriminators within the four categories for the COBOL reference group.

*Step V. Decide on a reference value (figure of merit) for each of the discriminating features*

The choice of a reasonable value for any specific complexity variable differs between programming environments.

TABLE VI.—Data Reference Complexity Rating for COBOL Reference Group

Program number	1	2	3	4	5	6	7	8	9	10	11	12	13	Average
Programmer	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>1</sub>	P <sub>4</sub>	P <sub>4</sub>	P <sub>3</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>4</sub>	P <sub>3</sub>	P <sub>5</sub>	P <sub>2</sub>	
# occurs	0	2	2	1	3	0	24	0	23	0	17	9	23	8
# redefines	0	0	11	2	0	0	6	0	23	2	2	0	1	3.6
Median span of var ref	4	10	15	12	28	22	4	9	4	12	2	8	14	11.1
Median dist bet var refs	0	13	16	7	25	10	9	24	15	31	7	4	24	14.5
# linkage var	3	3	4	4	9	5	6	3	4	11	0	11	5	5.2
Total # vars in procedure div	19	126	122	71	70	73	190	134	252	161	260	59	278	139.6
Total score	0	0	3	1	3	2	4	1	4	4	2	2	5	
Relative complexity rank for data ref	1	1	8	3	8	5	10	3	10	10	5	5	13	
Unique complexity rank	1	2	8	3	9	5	10	4	11	12	6	7	13	

TABLE VII.—Subprogram Interaction Complexity Rating for COBOL Reference Group

Program number	1	2	3	4	5	6	7	8	9	10	11	12	13	Average
Programmer	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>1</sub>	P <sub>4</sub>	P <sub>4</sub>	P <sub>3</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>4</sub>	P <sub>3</sub>	P <sub>5</sub>	P <sub>2</sub>	
Total # calls	1	8	<u>10</u>	<u>13</u>	6	8	<u>10</u>	<u>30</u>	<u>10</u>	<u>27</u>	0	0	8	10.1
R(# unique calls/ total calls)	<u>1</u>	<u>2</u>	<u>1</u>	<u>5</u>	<u>5</u>	<u>5</u>	<u>1</u>	<u>15</u>	<u>2</u>	<u>17</u>	-	-	<u>1</u>	.43
# params for PROC DIV	<u>3</u>	0	0	4	<u>6</u>	<u>5</u>	0	<u>3</u>	0	<u>8</u>	0	<u>4</u>	0	2.5
# ENTRY pts	<u>0</u>	<u>1</u>	<u>1</u>	0	<u>0</u>	<u>0</u>	<u>1</u>	<u>0</u>	<u>1</u>	<u>0</u>	0	<u>3</u>	<u>1</u>	.6
Avg # entry params	0	<u>3</u>	<u>2</u>	0	0	0	<u>3</u>	0	<u>4</u>	0	0	<u>1</u>	<u>5</u>	1.4
# changes in params	0	<u>3</u>	<u>1</u>	0	0	0	<u>1</u>	<u>1</u>	<u>3</u>	0	0	0	<u>1</u>	.8
Avg # params passed	<u>4</u>	3	4	3	<u>5</u>	<u>4</u>	<u>5</u>	<u>4</u>	3	<u>6</u>	0	0	<u>4</u>	3.5
# linkage vars	<u>0</u>	0	<u>13</u>	1	<u>8</u>	<u>3</u>	<u>12</u>	<u>0</u>	0	<u>11</u>	0	<u>14</u>	<u>0</u>	4.8
# copy in linkage	<u>3</u>	<u>3</u>	0	<u>3</u>	<u>3</u>	<u>2</u>	0	<u>3</u>	<u>4</u>	<u>2</u>	0	0	<u>5</u>	2.2
Total score	3	5	7	4	4	3	7	5	6	5	0	4	6	
Relative com- plexity rank for inter- action	2	7	12	4	4	2	12	7	10	7	1	4	10	
Unique com- plexity rank	2	7	12	4	5	3	13	8	10	9	1	6	11	

The purpose of the index is to produce a measure for a program within its own typical programming environment. The choice of a reference value for each discriminating characteristic was based solely on the source code within the reference group of programs.

Therefore, an average (and in cases where the range of values was too wide, a median) across a discriminating variable's values for each program in the reference group was chosen as the reference value against which programs would be evaluated (right-hand column in Tables V thru VIII). Since authors differed as to what program characteristics most affected complexity, the weighting given to each discriminating complexity characteristic was 1.0.

*Step VI. Assess a complexity score and index of complexity within each of the 4 complexity categories*

Essentially, the process of producing a complexity score is as follows: [K = Number of programs, N = Number of discriminating variables]. Each  $d_{ki}$  represents a measured value of the discriminating variable,  $v_i$ , for a program  $P_k$ . The measured value,  $d_{ki}$ , of each  $v_i$  for a program  $P_k$  is contained in the rows of Tables V thru VIII. Each  $w_i$  is the weighting factor for a discriminating variable  $v_i$ .

**Select a reference value:** (see Step V)

Compute an Average,  $A_i$ , where

$$A_i = \frac{\sum_{k=1}^K d_{ki}}{K}$$

for every  $i$ , where  $1 \leq i \leq N$ .

**Evaluate against the reference value:**

Compute a score  $u_{ki}$  where

$$u_{ki} = \begin{cases} 1 & \text{if } d_{ki} \geq A_i \\ 0 & \text{if } d_{ki} < A_i \end{cases}$$

The entries where  $u_{ki} = 1$  have been indicated by underlining the  $d_{ki}$  entries in Tables V thru VIII. Note that rounded values of the  $d_{ki}$  were used except in instances where  $d_{ki}$  had a majority of 0 as its measured value.

**Sum the scores with their respective weights:**

Assess a complexity score,  $S_k$ , for each program  $P_k$ .

$$S_k = \sum_{i=1}^N w_i u_{ki}$$



TABLE VIII.—Structure/Control Flow Complexity Rating for COBOL Reference Group

Program number	1	2	3	4	5	6	7	8	9	10	11	12	13	Average
Programmer	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>1</sub>	P <sub>4</sub>	P <sub>4</sub>	P <sub>3</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>4</sub>	P <sub>3</sub>	P <sub>5</sub>	P <sub>2</sub>	
$R(\frac{\text{verbs}}{\text{breaks}})$	-	5	5	8	7	11	5	5	3	17	4	4	5	6.1
# breaks in flow	0	<u>65</u>	<u>46</u>	17	25	11	<u>68</u>	<u>69</u>	<u>154</u>	20	<u>136</u>	<u>27</u>	<u>175</u>	62.5
Avg # instructions following IF	6	5	2	6	10	9	2	8	3	13	2	3	5	5.7
# outer IFS	<u>1</u>	<u>43</u>	<u>39</u>	<u>13</u>	<u>12</u>	<u>7</u>	<u>51</u>	<u>23</u>	<u>96</u>	<u>16</u>	<u>99</u>	11	<u>99</u>	39.2
% instrucs after IF (GO or PERFORM)	0	5	<u>21</u>	3	1	0	<u>53</u>	5	6	0	<u>25</u>	10	6	20.8
$R(\frac{\text{indep PERFORMS}}{\text{unique PERFORMS}})$	0	<u>2</u>	<u>5</u>	<u>4</u>	<u>6</u>	<u>6</u>	<u>1</u>	<u>3</u>	<u>4</u>	<u>6</u>	<u>8</u>	<u>2</u>	<u>2</u>	.4
Max breadth PERFORM nesting	0	<u>35</u>	1	2	2	1	0	4	<u>22</u>	2	<u>37</u>	3	<u>71</u>	13.8
Avg # paths through PERFORM	0	11	2	3	3	2	3	10	8	3	<u>205</u>	4	<u>21</u>	21.2
# loops	0	0	<u>6</u>	0	0	1	<u>26</u>	0	2	0	<u>26</u>	2	<u>1</u>	4.9
Span largest strongly conn region	0	0	<u>51</u>	-	-	1	<u>46</u>	-	<u>19</u>	-	4	2	<u>10</u>	10.2
# back branches	0	<u>41</u>	<u>1</u>	1	2	0	<u>0</u>	5	<u>48</u>	1	<u>32</u>	1	<u>67</u>	15.3
Avg span back branches	0	<u>192</u>	18	21	19	0	0	47	<u>255</u>	30	25	53	<u>472</u>	87.1
# down branches	2	<u>125</u>	93	57	84	33	123	<u>141</u>	<u>244</u>	121	<u>243</u>	53	<u>307</u>	125.3
Avg span of down branch	4	15	<u>30</u>	12	22	19	8	<u>47</u>	<u>25</u>	23	14	7	<u>86</u>	24
# loops composing largest str conn	0	0	<u>5</u>	0	0	1	<u>13</u>	0	1	0	<u>3</u>	1	1	1.8
Avg # exits out of a loop	0	0	<u>7.2</u>	0	0	1	<u>3.9</u>	0	3	0	<u>2.9</u>	1	<u>3</u>	1.7
# intersects bet loops	0	0	<u>10</u>	0	0	0	<u>44</u>	0	0	0	<u>25</u>	0	0	6.2
Total score	1	8	9	2	1	1	9	6	11	2	13	1	12	
Relative complexity rank	1	8	9	5	1	1	9	7	11	5	13	1	12	
Unique complexity rank	1	8	9	5	2	3	10	7	11	6	13	4	12	

*Step VII. Produce an overall figure of merit (index of complexity)*

The programs can now be ranked in order by their scores as indicated in the last 2 rows of Tables V thru VIII. This produces an overall index of complexity,  $R_k$ , within each complexity category.

The indices in each of the four categories provide a detailed complexity profile for a program. Figure 1 illustrates this with a histogram of the complexity indices within each category in order by programmer and for each programmer in order by his/her rating of each program. With respect to

the four complexity indices, COBOL programmer P<sub>4</sub> was inconsistent only in ranking what was a simple program on a medium scale. COBOL programmer P<sub>3</sub> ranked his programs in an order opposite to our overall results. However, the structural complexity assessed against his programs agrees with P<sub>3</sub>'s original ranking. Programmer P<sub>3</sub> did not take into account all aspects of complexity (e.g. interaction/interconnection) when evaluating his programs.

In turn, an overall figure of merit can be determined for the reference group of programs. For this COBOL data, the scores  $S_{kj}$  and rankings  $R_{kj}$  across the four complexity categories were averaged and used to assess an overall com-

plexity score  $S_k'$  normalized on a scale of 0 to 10. Each program  $P_k$  was in turn ranked according to this score producing an overall index of complexity  $I_k'$ .

Table IX illustrates the programmers' original ratings and rank versus the index of complexity.

UTILIZATION OF THE INDEX

The results of the index, through the set of variables found to be discriminators, provide a differentiation of programming styles within the COBOL reference group. When analyzing the complexity of 12 programs written by programmers  $P_1, P_2, P_3,$  and  $P_4$  (see Table IV) across the three programming styles defined in Table II, it was found that the controlled structured group produced far less complex programs than the other two groups. The average relative complexity rank of each group under the four categories of complexity was as follows:

	Controlled Structured	Uncontrolled Structured	Unstructured
Instruction Mix	3.7	10.7	8.3
Data Reference	5	8	7.7
Interaction/Interconnection	4.5	9	8.3
Structure/Control Flow	3.3	10.3	10.3

Structured programming is supposed to reduce the complexity of a program and the above results justify this within the controlled structured group. However, the supposedly structured (but controlled) programs produced the highest complexity ranking. The set of discriminating variables under each of the four categories were analyzed in an attempt to understand this.

An analysis of the complexity profiles found several data points which differentiated between the three programming styles and distinguished each from the other. Essentially, the discrimination between programming styles produced by the types of data points presented in Table II and discussed previously for a single program within each of the three categories were equally applicable across the 12 programs.

ADAPTING THE INDEX TO DIFFERENT PROGRAMMING ENVIRONMENTS

The technique of calculating an index of complexity as an evaluation of a program is quite flexible. The basic assumption is that programs should be judged by an objective set of program characteristics known to be relevant to complexity and should be judged within a reference group of

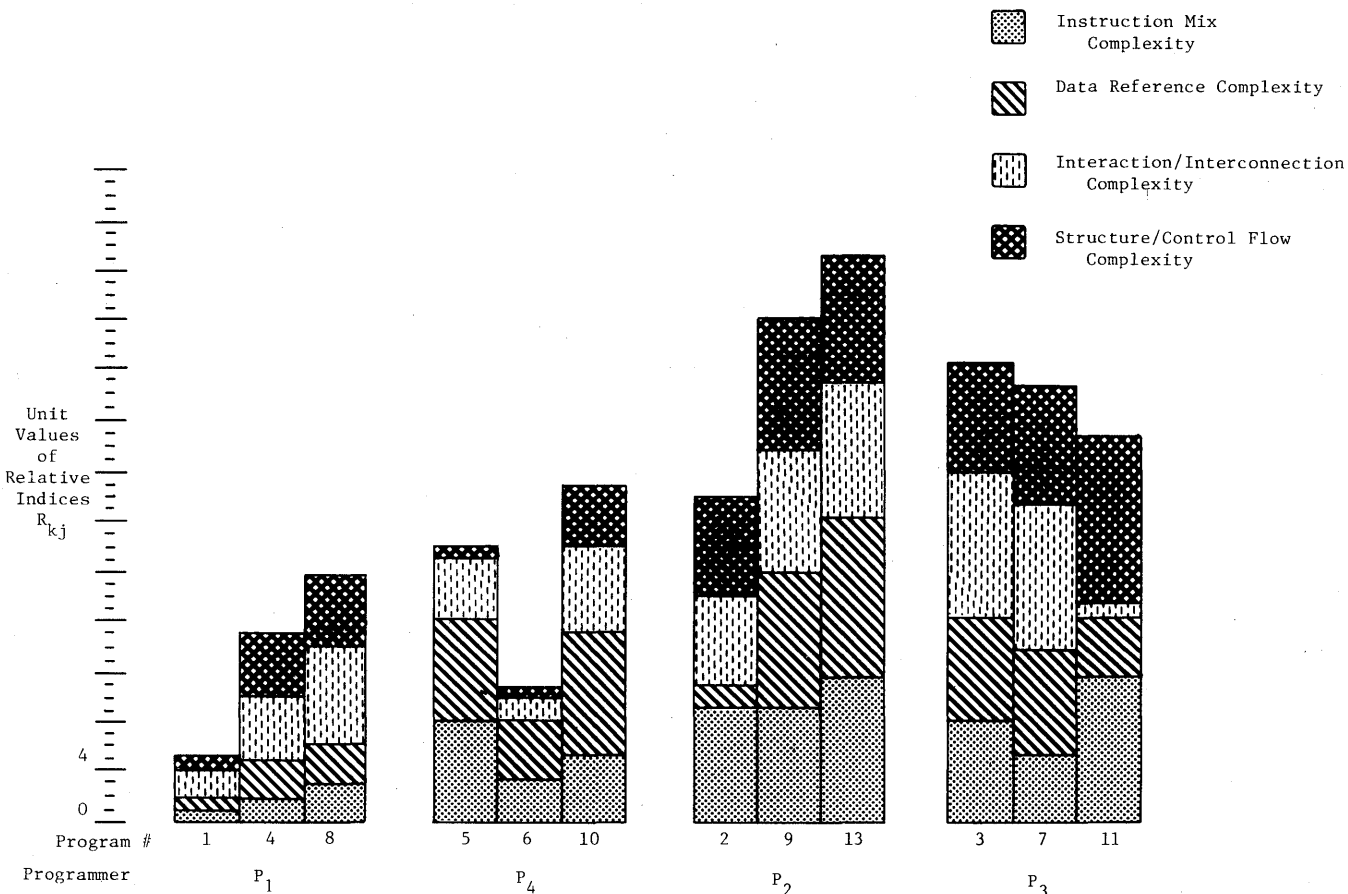


Figure 1—Histogram of relative complexity indices for the four complexity categories.

TABLE IX.—An Overall Index of Complexity

Program #	1	2	3	4	5	6	7	8	9	10	11	12	13
Overall Complexity Score ( $S_k^1$ )	.77	6.2	8.5	2.3	3.8	1.5	7.7	3.8	9.3	5.4	6.9	3.1	10.0
Index of Complexity ( $I_k^1$ )	1	8	11	3	5	2	10	5	12	7	9	4	13
Programmers' Rating	1	2	4	5	5	5	5	7	7	7	7	8	9
Relative Rank By Programmers' Rating	1	2	3	4	4	4	4	8	8	8	8	12	13

production programs reflective of their programming environment.

A manager can choose a set of complexity characteristics as a basis relevant to his/her software environment. The language of the programs to be measured is arbitrary. The technique for computing the index is independent of language and has been used on a set of FORTRAN programs.<sup>20</sup>

The static analysis, described previously, can be accomplished in large part by tools that should exist in most programming environments. The analysis done to produce a set of discriminating variables from the results of the static analysis involve only a careful examination of the data. Refinements in this analysis can be made as the reference group increases. Instead of a weighting of 1.0 across all discriminating characteristics, a manager can assess his/her own weights making the index more adaptable to opinions on the impact of specific program characteristics or the importance of structural complexity versus data reference complexity, etc.

The program profiles and complexity rankings have illustrated that there is a large amount of data that indeed differentiate between programs both within a language and across language usage. These results emphasize that there is not just one aspect of a COBOL program that sufficiently defines its complexity. A program has to be analyzed on all its merits and a manager must therefore make judgments on software accordingly.

## REFERENCES

- Webster's New Collegiate Dictionary, G & C Merriam Co., 1973.
- Gunning, R., *How to take the fog out of writing*, The Dartnell Corp., 1964.
- McCabe, T. J., "A complexity measure," *IEEE Transactions on Software Engineering* SE-2, 4 (December, 1976), pp. 308-320.
- Sullivan, J. E., *Measuring the complexity of computer software*, MITRE Corporation, MTR-2648, Vol. V, June, 1973.
- Myers, G. T., "An extension to the cyclomatic measure of program complexity," *SIGPLAN Notices* 12, 10 (October, 1977), pp. 61-64.
- Hansen, W. J., "Measurement of Program Complexity by the Pair (Cyclomatic Number, Operator Count)," *SIGPLAN Notices* 13, 2 (March, 1978), pp. 29-33.
- Peterson, W. W., Kasami, T. and Tokura, N., "On the capabilities of while, repeat, and exit statements," *CACM* 16, 11 (November, 1973), pp. 503-512.
- Woodward, M. R., et al., "A measure of control flow complexity in program text," *IEEE Transactions on Software Engineering*, Vol. SE-5, No. 1, January, 1979, pp. 45-50.
- Cobb, G. W., "A measurement of structure for unstructured programming languages," *Proceedings of the Software Quality and Assurance Workshop*, November, 1978, pp. 140-147.
- Gilb, T., *Software Metrics*, Winthrop Publishers, Inc., 1977.
- McClure, C. L., "A Model for Program Complexity Analysis," *Proceedings of 3rd International Conference on Software Engineering*, IEEE Cat. No. 78CH1317-7C, pp. 149-157.
- Chapin, N., "A measure of software complexity," *Proceedings of the National Computer Conference*, 1979, pp. 995-1002.
- Chen, E. T., "Program Complexity and Programmer Productivity," *IEEE Transactions on Software Engineering*, Vol. SE-4, No. 3, May, 1978, pp. 187-193.
- Halstead, M. H., *Elements of Software Science*, Elsevier North-Holland, Inc., N.Y., 1977.
- Shooman, M. and Laemmel, A., "Statistical Theory of Computer Programs—Information Content and Complexity," *Digest of Papers COMPCON Fall 77*, IEEE Cat. No. 77CH1258-3C, pp. 341-347.
- Zolnowski, J. C. and Simmons, D. B., "Measuring Program Complexity," *Digest of Papers of Fall COMPCON 77*, IEEE Catalog No. 77CH1258-3C, pp. 336-340.
- Thayer, T. A., et al., *Software reliability study*, RAD-TR-76-238, August, 1976.
- Knuth, D. E., *An empirical study of FORTRAN programs*, U.S. Government Report AD-715-513, February, 1971.
- Aho, A. Y. and Ullman, J. D., *The theory of parsing, translation, and compiling*, Vol. 11: *Compiling*, Prentice-Hall, Inc., 1973.
- Zolnowski, J. C. and Simmons, D. B., "A complexity measure applied to FORTRAN," *Proceedings of COMPSAC 77*, IEEE Catalog No. 77CH1291-4C, pp. 133-141.
- Mills, H. D., "The complexity programs" in *Program Test Methods*, ed. W. D. Hetzel, Prentice-Hall, Inc., 1973, pp. 225-239.
- Zolnowski, J. C., "A Composite Measure of Program Complexity," AUERBACH Computer Programming Management series, 1978.

# The complexity of an individual program

by JOHN L. MCTAP  
Software Engineer  
Menlo Park, California

## NEED

Zolnowski and Simmons proposed an interesting measure of program complexity.<sup>1,2,3</sup> But it suffers from four major practical defects:

- a) It cannot be applied to an individual program without first identifying a group of existing programs as a reference base.
- b) It does not provide a reliable measure since the measure usually changes each time the composition of the group of programs changes.
- c) It is burdensome to compute because the computation must be for an entire group of programs to make possible measuring the complexity of an individual program.
- d) It cannot be applied to the components within a program, such as modules or subroutines.

In spite of these deficiencies, the measure has some attractive strengths. An example is the use of program features.<sup>1</sup> These strengths could be enhanced. Ideally, these enhancements should be attained while eliminating the practical defects. Then, the enhancements would also be improvements.

## IMPROVEMENT

The features of a program that can qualify for use in the measurement of program complexity have been specified in terms of criteria by Zolnowski and Simmons.<sup>1</sup> Briefly, the criteria for acceptable features are:

- 1—The features must have been identified by some authority as being relevant to increasing or decreasing program complexity.
- 2—The features must exist in some degree or amount in a program.
- 3—The features must be expressible and measurable in quantitative units.
- 4—The features must be commonly found in some degree in nearly all programs to be included in the group.
- 5—The available documentation must provide written evidence of the features.

Some examples of program features often meeting the criteria are: the total number of imperative instructions, the average depth of the "IF" nesting, the total number of lines

of source code, the total number of entry points, the total number of Boolean variables declared, the average number of parameters passed, the average number of lines of source code jumped by a forward transfer of control, and the total number of lines of annotation in the source code.

As an improvement on the Zolnowski and Simmons measure, it is here suggested that a sixth criterion be added.

6—The expected amount of the features must be cited in the organization's local standards manual. And a change is here suggested in the fourth criterion, by dropping the phrase "in the group."

These suggestions effectively eliminate the need to use a group of programs. Instead, the local standard effectively substitutes for the group. Whereas the selection of programs to comprise the group may be haphazard or limited, the selection of local standards is commonly given much thought, careful attention, and extensive review. Also, local standards are normally promulgated to encourage good practices and serve as a basis for assessing the application of those practices. In contrast, existing programs only by chance may be exemplary, and at the best, only reflect the *past* practices of personnel.

The reason why Zolnowski and Simmons have proposed the use of a group of programs is simple: the statistical law of large numbers. As sample size increases, the character of the sample is more likely to be the same as the character of the group the sample was drawn from. If you want to know how complex programs are at some user organization, sample them for an estimate. Then contrast that estimate against any program which may interest you from that organization. This makes the Zolnowski and Simmons measure of complexity really a measure of the deviance or difference from past programming practices.

Applying the law of large numbers to the record of the past sometimes offers cold comfort for the present. A present program may appear deviant, yet the deviance may be deliberate to correct undesirable past practices. An example might be the use in COBOL of "PERFORM" within "PERFORM." Further, past practices can hide from attention complexity-raising present practices. An example may be the use of a declared variable for multiple purposes (it is one way to reduce the total of the data declarations).

In many organizations, present programming practices are governed by the current local standards manual. This usually

describes the target or acceptable practices for the organization. For example in a COBOL environment, a standards manual might specify that IF's may be nested provided the nesting depth does not exceed three. The suggestion here is that the target or acceptable practices substitute for the actual past practices proposed by Zolnowski and Simmons. To support this substitution of local standards for the characteristics of the group of programs, a simplification can be made in the computational procedure. This is described below.

This substitution puts attention on a lightly treated aspect of the Zolnowski and Simmons measure, the selection of weights. Uniform weighting is done in the Zolnowski and Simmons measure. Here with the improvement suggested, the opportunity exists to weight features differentially to reflect the local conditions, or the judgments of the local management, or the recommendations of the accepted authorities about the relative importance or priority of the features.

## COMPUTATION

The computation of the improved measure of program complexity has two phases. One phase works from the local standards, and prepares for the second phase. The second phase works from any individual program, and results in a measure of the complexity of that program. No group of programs is needed.

The first step in the first phase is to select the features to use in the comparison. Each of these features must meet all six criteria cited previously. A decision will be needed on the relative weight or importance in program complexity for each feature. A selection will also be needed of which direction in the measure for the feature indicates high complexity—that is, is a low number or a high number an indicator of high complexity for the feature? Some features, for example, the average number of parameters passed, increase complexity when large, and decrease complexity when small. Some other features operate in the reverse direction, the authorities indicate.

Second, for each of the features selected, a reference value must be chosen. This is the amount against which the comparison will later be made. Where this level should be set is a matter for local determination and the application of the six criteria noted earlier. As Zolnowski and Simmons have pointed out, existing programs may fruitfully contribute some evidence as to current practices. But differential studies as to successful versus unsuccessful software could yield far better evidence for setting the local reference level of features, than just a raw sampling of past programs.

Third, once the reference levels have been established, they must be expressed in the reference vector  $R$ , with one element for each feature.

Fourth, to accompany this reference vector, a vector of weights  $W$  is needed, with one element for each element in the reference vector. That is, the weights in the weight vector

represent the size of the contribution to program complexity of the feature.

Fifth, a direction vector  $D$  must be prepared, with one element for each feature, to match the reference and weight vectors. But the direction vector is all +1 or -1 elements. A +1 element indicates that an increase in the amount of the feature contributes to simplicity, and a -1 to complexity.

The above five steps of the first phase must be done before attempting the second phase to measure the complexity of individual programs. But once the first phase is done, any number of individual programs may be measured by successively applying the steps of the second phase. The second phase has the following steps.

A brief example can illustrate the steps in the second phase, as shown in Figure 1. Assume that phase one has given the list of features, the reference vector  $R$ , the direction vector  $D$ , and the weight vector  $W$  (see Figure 1 top). The number and selection of features deliberately has been made simple and small here for convenience. Figure 1 bottom shows the second phase step-by-step computational procedure to get the program complexity score.

First, the documentation of the program is examined for evidence of each feature. Evidence found is interpreted in numeric form and is recorded feature by feature as the feature vector  $F$  of this program.

Second, the magnitude ( $M$ ) of the difference between the feature vector and the reference vector is determined. A simple subtraction takes care of this matter, but care must be taken for which to use as the subtrahend and which as the minuend in order to match the direction vector. If the subtraction is of the feature vector  $F$  from the reference vector  $R$ , then a positive sign on an element in the magnitude vector means that the reference vector asked for more of the feature than the feature vector reported the program to have.

Third, element by element, the difference vector  $M$  from the prior step is multiplied by the direction vector  $D$ , to alter the signs and yield a product vector  $P$ . Now in the product vector  $P$ , a plus indicates undesirable high complexity and a minus desirable low complexity.

Fourth, a change vector  $C$  is created from the product vector  $P$ . Whenever the product vector element sign is a minus, a zero is put in the change vector  $C$ . Whenever the product vector element is a plus, a +1 is put in the change vector  $C$ .

Fifth, the score vector  $S$  is obtained by multiplying the change vector  $C$  by the weight vector  $W$ , element by element.

Sixth, the sum of the elements in the score vector is the complexity score for the program.

The score is independent of any particular group of past or present programs. The score reflects deviations from the local standard. Further, it reflects those aspects of the programs which are known to be significant contributors to complexity. This has been achieved in a manner which makes the score relevant to a particular organization's practices, and permits an easy comparison between programs subject to that standard. This also permits a simple comparison of the effects of proposed or actual changes in standards.

PHASE ONE

<u>Feature Identification</u>		<u>Vectors</u>		
ID	Description	W	D	R
A	Average verbs per module	2.0	-1	10
B	Ratio of PERFORM verbs	1.5	-1	0.1
G	Ratio of IF	1.2	-1	0.05
H	Ratio of MOVE verbs	1.0	+1	0.33
E	Average variables per module	1.8	-1	8
J	Maximum IF nesting depth	1.5	-1	4

PHASE TWO

ID	R	-	F	→	M	×	D	→	P	→	C	×	W	→	S
A	10		8		+2		-1		-2		0		2.0		0.0
B	0.1		0.18		-0.08		-1		+0.08		1		1.5		1.5
G	0.05		0.08		-0.03		-1		+0.03		1		1.2		1.2
H	0.33		0.31		+0.02		+1		+0.02		1		1.0		1.0
E	8		9		-1		-1		+1		1		1.8		1.8
J	4		3		+1		-1		-1		0		1.5		<u>0.0</u>

Program complexity score 5.5

Figure 1—Example of the computation of the complexity score for a small modularized COBOL program using a limited list of features

## EXTENSION

Further, the technique can be extended from the program to the subroutines or the modules individually within the program. Zolnowski and Simmons have opposed this, apparently on the grounds of statistical instability from the expected smaller sample sizes. But with the improvements presented in the previous section, such an extension is both reasonable and desirable without the need to identify subsets of modules.

Complexity in the program must arise from the features

of the program, as Zolnowski and Simmons have pointed out. Yet, except for a few features, such as overall program size, the features of the program exist because either 1—they are also features of the program's component modules, or 2—they can be easily redefined to be features recognizable at the module level. For example, the count of the number of COBOL "IF's" in the program occurs in the modules. And, for example, the percentage of the total count of the verbs used in a COBOL program that are "PERFORM" verbs can be redefined for module complexity to be the percentage within the module. The results of a computation of the complexity of the modules of the sample program is

shown in Figure #2. Note that the program complexity is not necessarily the average of the module complexity.

This extension is valuable in the use of the complexity measure for efforts to reduce or control complexity. If a measure is global to the program, then the use of the feature must be located or found *in* the program in order to modify the use of the feature. Yet, most of the time, the feature has been used in specific modules. It is far less time consuming to search a few easily identified modules for the presence of a feature than to search an entire program. And, modification of the complexity in the modules nearly always results in a corresponding modification of the program complexity.

## DISCUSSION

A matter of concern for any measure of program complexity is its validity. The validity of the proposed measure is at least as high as that of the Zolnowski and Simmons measure because the five selection criteria are the same for both in selecting the features. The added sixth criterion offers an improvement in validity because it allows substituting a standard which can be carefully conceived for the haphazard chance of a group of programs. If the standard is haphazard or ill-conceived, the proposed measure is still as good as the Zolnowski and Simmons measure, as long as the other five criteria are observed.

The proposed measure is more responsive to local concerns. For example, factors that are relevant for measuring

the complexity of real-time software may be, in the experience of an organization, significantly different from those measures which reflect the complexity of delayed-time software. These can be reflected explicitly in the measure proposed without the need to accumulate two groups of programs. In addition, specific characteristics related to the particular hardware configuration, software resources, and local practices are included, provided the criteria are satisfied.

Better flexibility and adaptability are offered by the proposed measure. The proposed improvement makes possible an easier comparison of work done by different teams or by different individuals on the same or different teams. It makes possible an easier comparison of the professional growth of individuals in their work assignments. It makes possible an evaluation and comparison of the software work done at one site with that done at another. It makes possible a quantitative measure to be developed of what distinguishes successful software from unsatisfactory software. All these benefits require that a relevant standards manual exist.

The proposed measure makes possible the easy revision or modification of the feature list. It is common experience that when some people know that the quality or quantity of their performance is being measured, those people take action to make their work appear good, without actually improving their work. That is, people sometimes try "to beat the system." To retain validity, the "system" must change, and the proposed measure can be changed at a low overhead cost.

Feature	Module												Program
	F Vectors and Complexity Scores												
ID	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	
A	12	5	12	10	19	1	22	2	1	2	2	1	8
B	17	0	33	40	0	0	27	0	0	0	0	0	18
G	17	0	8	0	0	0	18	0	0	0	0	0	8
H	42	60	33	20	32	0	32	0	0	0	0	0	31
E	9	11	9	7	28	3	23	5	3	5	5	3	9
J	1	0	1	0	0	0	3	0	0	0	0	0	3
Score	6.5	0.0	6.5	2.5	4.8	1.0	7.5	1.0	1.0	1.0	1.0	1.0	5.5

Note: The program F vector is not the arithmetic average of the module F vectors. For that reason, the program complexity score is not the arithmetic average of the module complexity score. Module and program complexities are calculated in the same way--i. e., modules are treated like individual programs.

Figure 2—Results of applying the same computational procedure to the modules of the program shown in Figure 1

Further, when people do actually improve their work, the variation in their work often becomes smaller. Then, the units of measure used in the past may become inappropriately large. The proposed measure of complexity can easily compensate for this, and retain its validity just by changing the reference vector and the units of measure for both the reference and feature vectors—something partly or fully lost in using a group of programs. This permits an orderly growth in the focusing of the organization's attention upon those aspects of design and implementation practices which contribute to complexity. To the extent these are brought under control, attention can then shift to new factors as they become dominant. These in turn can be incorporated until they, too, are better controlled, provided, of course, that they meet the six criteria.

The reliability of the improved measure is as high or higher than that of the Zolnowski and Simmons measure. All the aspects of that measure which contributed to reliability have been preserved. Further, the computational procedure has been simplified, freed from a tie to a group of programs, and made less burdensome, reducing several sources of unreliability.

## CONCLUSION

A comparison of the proposed measure has been presented by Chapin.<sup>4</sup> Compared to other measures, the validity and reliability of the proposed measure is at least as good as that of the Zolnowski and Simmons measure.<sup>4</sup> The cyclomatic number advocated by McCabe lacks the breadth of the proposed measure.<sup>5</sup> The McClure measure has a more laborious computational procedure, and requires a well-structured design.<sup>6</sup> Yet the use of structured techniques does change the features of programs, as Elshoff<sup>7</sup> and Zolnowski<sup>3</sup> have documented.

By adding a sixth criterion, the proposed improvement effectively subsets the features covered in the Zolnowski and Simmons measure to those covered by local standards. This subsetting enhances the validity by eliminating the locally extraneous and irrelevant features, and improves the sensitivity and responsiveness of the proposed measure. This subsetting also converts the measure into a comparison against a desirable objective or target, instead of against a melange of past practices.

Finally, the most significant aspect of the proposed improved measure of program complexity is its applicability to individual programs and to the modules within the programs without suffering from the ever-shifting membership of the group and free from the burden of computation for a group of programs. Programs and modules can, with the proposed improved measure, be rated standing alone for complexity, in quantitative and reliable terms.

## REFERENCES

1. Zolnowski, Jean C. and Simmons, Dick B., "A complexity measure applied to FORTRAN," *Proceedings of the COMPSAC 1977* (Long Beach, CA: IEEE Computer Society, 1977), pp. 133-141.
2. Zolnowski, Jean C. and Simmons, Dick B., "Measuring program complexity," *Proceedings of the 1977 Fall COMPCOM* (Long Beach, CA: IEEE Computer Society, 1977), pp. 336-340.
3. Zolnowski, Jean C. and Simmons, Dick B., "Measuring program complexity in a COBOL environment," a paper located elsewhere in these *Proceedings*.
4. Chapin, Ned., "A measure of software complexity," *Proceedings of the 1979 NCC* (Montvale, NJ: AFIPS, 1979), pp. 995-1002.
5. McCabe, Thomas J., "A complexity measure," *Software Engineering*, Volume SE-2, Number 4 (December 1976), pp. 308-320.
6. McClure, Carma L., *Reducing COBOL Complexity through Structured Programming* (New York: Van Nostrand Reinhold, 1978), 192 pp.
7. Elshoff, James L., "The influence of structured programming on PL/I program profiles," *Software Engineering*, Volume SE-3, Number 5 (September 1977), pp. 364-368.





# An information theory based complexity measure

by ELI BERLINGER

Nassau Community College  
Garden City, New York

## INTRODUCTION

There have been numerous measures proposed to measure program complexity. Some are completely heuristic, comparing certain measurable program features against a set of predefined standards. Some are topological, based on the number of regions on the control or data graph of the programs or a combination of the above, and of course, there is Halstead's Software Physics.

All of these measures have their deficiencies and, no doubt, so will ours. We have, however, set ourselves the goal of eliminating some of them and to provide a measure which has mathematical and intuitive correctness and which will have a good correlation with observed facts.

### Goals

A complexity measure based on all elements of the program. Everything can produce an error, even a simple assignment statement or MOVE instruction. They should all have some effect on the measure. This requirement is not met in the topological measures, for instance.

To satisfy both intuitive desires and psychological realities, we want the measure to be more sensitive to infrequently used elements of the language being used than to commonly used ones. It is reasonable to suppose one can do more accurately what one does more frequently.

A particular element of a language used in a complex manner should contribute more to a complexity measure than the same element used in a simpler manner. For example, a nested loop would seem to be more complex than a sequence of two simple loops. Few of the measures seen to date have adequately addressed this situation.

The measure should allow for automated techniques in its calculation.

These are our primary goals. As will be seen later, additional advantages will follow as a direct result of the measure definition.

## STATE OF THE ART

A number of complexity measures have been proposed. A review of some of them will be presented in this section,

together with a brief critique of some of their strengths and weaknesses. As will be noted, a number of the measures seem to be completely heuristic with no apparent mathematical justification.

Hellerman<sup>1</sup> proposed a measure having some relation to information theory. His measure is based entirely on the number of inputs and outputs. Let  $X$  be the domain of inputs and  $Y$  be the range of outputs. Each element,  $Y_i$ , of the range is the output of a class of inputs,  $X_i$ . The  $X_i$  form a mutually exclusive set of  $N$  classes. If we conceive of a program as reducible to a table lookup, let  $|X_i|$  be the number of elements of  $X_i$  and  $|X|$  be the number of elements of  $X$ . Then each  $Y_i$  appears  $|X_i|$  times. The probability of its occurrence is  $P(Y_i) = |X_i|/|X|$  and its information content is  $-\log_2 |X_i|/|X| = \log_2 |X|/|X_i|$ . The total information content,  $w(f)$  which Hellerman calls the "computational work" is  $w(f) = \sum_{i=1}^N |X_i| \log_2 |X|/|X_i| = |X| H(P_1, \dots, P_N)$  where  $H$  is the entropy function.

There are several objections to the use of Hellerman's measure. Two will be given.

First, it is necessary to know precisely how many input values yield a specific output value. In most programs, the domain input values is an enormous set, as is the range of output values. In all likelihood, neither domain, nor range, will ever be exhausted.

Second, even assuming all elements of the domain can be used, we would require the computer to evaluate all elements of the range to determine the domain classes. This means that every input value has to be used to obtain the output values, which will determine the domain classes, which can then yield the measure. By this time, the program is completely debugged and delivered and we no longer need the measure.

McCabe<sup>2</sup> proposed a measure which is easily obtained at an early stage. Known as the "cyclomatic number," it is based on the number of linearly independent paths through the control graph of the program. Let  $N$  be the number of nodes,  $E$  the number of edges, and  $P$  the number of connected components. Then the cyclomatic number,  $V$ , is  $V = E - N + 2P$ .

McCabe derives a number of simplifications for the calculation of  $V$ . Using Euler's formula,  $N - E + R = 2$ , where  $R$  is the number of regions of a plane control graph, we get  $R = E - N + 2$ . Therefore, if there is only one connected com-

ponent,  $R=V$ . In words,  $V$  is the number of regions on a plane control graph, counting the external region.

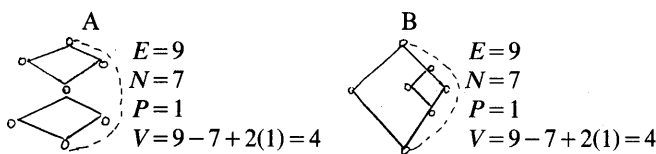
Using a result of Mills,<sup>3</sup> McCabe achieves another simplification. If  $\pi$  is the number of predicates in the program, then  $V=\pi+1$ . Therefore, to calculate  $V$ , one need only count the number of binary conditions contained in IF's and DO's to calculate  $V$ .

Myers<sup>4</sup> proposes a modification to McCabe's cyclomatic measure. He shows that difficulties arise if one calculates  $V$  by counting the number of predicates (conditions) and that other difficulties arise if one counts only the number of decision statements, IF's and DO's. He proposes to make  $V$  an interval whose lower bound is the number of decisions + 1, and whose upper bound is the number of predicates + 1. By using an interval, Myers removes the question of whether or not the statements

IF (A & B) THEN ...  
                                   ELSE ... ; are more  
 complex than IF A THEN IF B THEN ...;  
                                   ELSE ...;  
                                   ELSE :

The cyclomatic measure is a good one. It is easily obtained and comparisons between programs are possible. However, two programs consisting only of sequences will both have  $V=1$  regardless of the number of statements or variables contained in the program. The measure is sensitive only to branching or looping. Nothing else which might contribute to the complexity of a program is considered. Some purely sequential programs can be quite complex, however, and their measures should reflect this.

A much more serious problem appears to exist. A sequence of IF-THEN-ELSE's has a higher cyclomatic number than a nest. Consider the following two diagrams:



Both control graphs have a cyclomatic measure of 4, yet, intuitively, a nest should be more complex than a sequence since several predicates are operative simultaneously.

McTap<sup>5</sup> proposes a measure which compares certain measurable features of a program to a set of predetermined standard values. The greater the number of features which deviate from the norm on the side of complexity, the larger the measure. The measure is calculated in two phases, as follows:

#### Phase 1

1. Select those features, which are to be measured. The features must be "normalized" so that a given value means the same thing in all programs or modules. For example, the

number of decisions is not a satisfactory feature, while the proportion of decisions, compared to all statements, is.

2. Select, arbitrarily or by experience, a reference level for each feature.

3. Form reference vector,  $R$ , comprised of these levels.

4. Form a direction vector,  $D$ , where  $d_i = +1$  means the feature contributes to simplicity and  $d_i = -1$  means it adds to complexity.  $+1$  and  $-1$  are the only values used in  $D$ .

5. Form a vector,  $W$ , of weights for each feature.

#### Phase 2

1. Form feature vector,  $F$ , by studying the documentation and measuring each feature.

2. Form the magnitude vector  $M=F-R$ . Here, if the feature exceeds the reference level,  $m_i > 0$ , else  $m_i \leq 0$ .

3. Form product vector  $P=M*D$ . If  $P_i > 0$ , the  $i^{\text{th}}$  feature in the program contributes less complexity than the reference level. If  $P_i < 0$ , it contributes to greater complexity.

4. Form change vector,  $C$ :

$$C_i = \begin{cases} 0 & \text{if } P_i \geq 0 \\ 1 & \text{if } P_i < 0 \end{cases}$$

That is,  $C_i = 1$  for each feature which contributes more complexity than the reference level.

5. Form the score vector,  $S=C*W$ . If particular feature of the program is more complex than the reference value, the corresponding score vector element is the weight assigned to that feature. If not, the element is zero.

6. Form the complexity measure  $M=\sum s_i$ , that is, the measure is the sum of the elements of the score vector.

From a theoretical viewpoint, there is absolutely no mathematical correspondence between this measure and the number of bugs, or any other factor of importance. From a production viewpoint, this criticism is not important. If it has been experimentally determined that programs whose measure exceeds a certain value tend to be poorly written, then the measure can be used to detect such faulty programs so that they can be rewritten at an early stage of production. For that purpose, the measure appears suitable as a guide.

The measure will provide no correlation between the complexity measure and the number of bugs. The elements of the score vector are only two-valued. If the program feature is below the reference level, the corresponding element is zero. If above, it is equal to the weight. It would seem that the measure might be improved by redefining the change vector,  $C$  as follows:

$$C_i = \begin{cases} 0 & \text{if } P_i \geq 0 \\ -P_i & \text{if } P_i < 0 \end{cases}$$

The correlation between the measure and the number of bugs should now be considerably improved.

Chapin<sup>6</sup> proposes a measure which is based on the role played by the variables in each module. He considers four

types of variables:

- P*—input data needed for processing
- M*—output data assigned a value
- C*—data used as control (in loops, decisions, etc.)
- T*—data which remains unchanged in the module (passes through)

An I-O table is constructed for each module on which is indicated the role of each I-O variable. (Here, an I-O variable is any non-local to the module.) The measure is calculated as follows:

For each module;

1. Count, in the I-O table, the number of items in *C*, *P*, or *T* roles (output).
2. Multiply each by a weighting factor; 3 for *C*, 2 for *M*, 1 for *P*,  $\frac{1}{2}$  for *T*.
3. Sum the weighted counts.
4. Initialize *E* to 0 for all modules.
5. If a module serves in the exit test of an iteration then add to its *E* measure as follows: (a) if the control data item for the exit test comes from within this module, add 0 to *E*; (b) if it comes from the loop body, add 1 to *E*; (c) if it comes from outside the loop, add 2 to *E*. For example, if the control data is initialized outside the loop and modified within the loop, add 3 to *E*.
6. Convert *E* into a repetition factor, *R*, by adding 1 to the square of  $\frac{1}{3}$  of *E*:  $R = (\frac{1}{3}E)^2 + 1$ .
7. Multiply the sum of the weighted counts from step 3 by the modules respective *R*s.
8. Find the square root of the products from step 7. This is *Q*, the index of module complexity.
9. Calculate the program *Q* by finding the mean of the module *Q*s.
10. Calculate the system *Q* by finding the mean of the component module *Q* within the component programs.

It is felt that this measure has a number of disadvantages:

1. While the measure is based on the set theoretic definition of a function, some of the specific calculations appear to be experimentally derived. For example, step 6 requires the square of one third of the *E* value. Why this particular value is used is not theoretically clear.
2. It requires a detailed I-O table or equivalent for each module specifying precisely how each variable is to be used in the module. This may be desirable in some cases. A clear idea of the meaning of each data item is certainly essential. However, it would require close supervision to ensure that each variable is properly identified.
3. The calculation of the program and system *Q*s leaves much to be desired; the use of the mean has the effect of making large programs with many modules no more complex than an individual module. This does not seem to be desirable. As a minimum, the program and system *Q*s should be some additive function of the module *Q*s.

Chen<sup>7</sup> defines a measure on the topological properties of a graph. Given a "strongly connected proper flowchart," one which contains no dividing edges or bridges, nor weakly

connected subparts, he defines the "maximal intersect number," MIN, as the maximum number of edges which can be intersected by a continuous line drawn so as never to enter any region, including the external region, more than once.

If the graph of the flowchart is made up of serially connected subparts, the MIN =  $\sum$ MIN of subparts - (2  $\times$  number of subparts) + 2, for which he gives a proof.

The topological attribute MIN can be given analytically by the expression  $MIN = Z_n + 1$  where

$$Z_n = 1 + \sum_{i=1}^{n-1} \log_2(p_i X + q_i), X = 2$$

where *n* is the number of decision symbols on the flowchart or graph,  $q_i$  = probability that the *i* + 1<sup>th</sup> IF symbol is forming a serial relation with any of its preceding and adjoined IF symbols, and  $p_i = 1 - q_i$ .

For a given proper flowchart,  $q_i$  is either 1 or 0 depending on whether or not it is in the specified serial relationship.

$Z_n$  is called the "control structure entropy." The programmer is considered a source emitting IF symbols and  $Z_n$  is defined as the entropy of the "source" when it emits *n* IF symbols in a specified manner.

For a given total number of IF symbols, the programming job is to determine the exact flowchart structure. This will determine the  $p_i$ s and  $q_i$ s. Since there are many different alternatives which are seemingly equal, we can, a priori, set  $p_i = q_i = \frac{1}{2}$ . This gives  $Z_n = 1 + \sum_{i=1}^{n-1} \log_2(1 + \frac{1}{2}) = (n-1) \log_2 3 - n + 2 = Z_n$ . Given the number of IF symbols a program is to have,  $Z_n$  can be calculated since it depends only upon *n*.

Chen now derives a relationship between  $Z_n$  and *r*, the productivity in source statements per busy hour. The relationship is heuristic, derived to fit his observed points and is

$$r(Z_n) = \frac{1}{b/a + (c - b/a) \cdot e^{-a Z_n}}$$

where  $a = 0.3187$ ,  $b = 9.2451 \times 10^{-1}$  and  $0.7695 \times 10^{-1} \leq c \leq 0.7714 \times 10^{-1}$ .

For a given flowchart the measure  $Z_n$  has the property that any number of serially connected IF symbols do not contribute to complexity. Only if the IF symbols are "nested," will the complexity increase. While it is true that serially connected IFs seem to be inherently less complex than other connections, yet it seems to be an oversimplification to ignore them completely.

Using the "a priori" formula improves this situation somewhat. Here, each IF statement add  $\log_2 3 - 1$  to the entropy  $Z_n$ .

It should be noted that Chen's maximal intercept number is quite similar to McCabe's measure when there is only one connected component.

Mohanty<sup>8</sup> proposes the Entropy Loading measure based on the amount of information shared between modules or subsystems.

Assume *A* and *B* are two subsystems of *S* such that  $A \cap B = \phi$ ,  $A \cup B = S$ . A table is made with every subsystem

of  $A$  and  $B$  along one dimension and every system attribute along the other. An entry is 1 if a subsystem makes an assumption about the attribute, 0 otherwise. For each subsystem of  $S(A$  or  $B)$  we consider the submatrix consisting of attributes which have a nonzero entry for that subsystem. Applying Schutt's Entropy Metric<sup>9</sup> to each submatrix, we get  $H(A)$  and  $H(B)$ . If we calculate the Entropy Metric for the entire system  $S$ , we get  $H(S)$ . The Entropy Loading Measure is now defined as

$$C(S) = H(A) + H(B) - H(S).$$

It is a measure of the information shared between the subsystems  $A$  and  $B$ .

This measure is useful at the system design stage where the shared attributes are known, or, at least, hinted at. It provides no help in determining the complexity of a given module or subsystem, which would, however, be desirable at a later stage in program development.

Schutt<sup>9</sup> proposes the Entropy Metric. It is similar to Helder's Work Function<sup>1</sup> but is divided by  $|X|$ , the number of input classes. The definition of the Entropy Metric is

$$H = \sum \frac{|X_i|}{|X|} \log_2 \frac{|X|}{|X_i|}.$$

Halstead<sup>10</sup> has proposed a rather comprehensive theory in which he not only proposes a measure to correlate with the bugs in a program, he also attempts to predict this measure at the most primitive state of the programming when all that is known is the number of input and output variables.

Let  $\eta_1$  = no. of operators  
 $\eta_2$  = no. of operands  
 $N_1$  = total usage of operators  
 $N_2$  = total usage of operands  
 and  $N = N_1 + N_2$  be the total length of the program  
 then  $N = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$  and is called the program length.

While this relationship is interesting and Halstead demonstrates that it gives correct results for a number of programs, it is not the measure he uses.

He defines program volume,  $V$ , as

$$V = N \log_2 \eta \text{ where } \eta = \eta_1 + \eta_2$$

The potential volume,  $V^*$ , is the volume a program would have if a function existed which would solve the problem. There would be two operators; one an assignment statement, the other a call to the function. The potential volume is therefore:

$$V^* = (2 + \eta_2^*) \log_2 (2 + \eta_2^*)$$

where  $\eta_2^*$  is the number of input/output variables, each used once in the function parameter list.

The program level,  $L$ , is now defined as the ratio of the potential volume to the program volume:

$$L = \frac{V^*}{V}$$

The larger the volume of the program, the lower the level of its coding.

The program effort,  $E$ , is defined to be proportional to the volume and inversely proportional to the level:  $E = V/L$ . Here it is assumed that for a given volume, it is more difficult to comprehend a program written at a low level than at a high one, perhaps because of increased complexity for each statement, and for two programs written at the same level (remember that the level is unitless, being a ratio of like quantities) it is more difficult to comprehend the "larger" one.

It is the effort,  $E$ , which is used as the measure to correlate with the number of bugs in the program.

Halstead defines another measure,  $I$ , called the information content, defined  $I = LV$ . This measure, Halstead feels, correlates best with total programming and debugging time.

Halstead has given a comprehensive and, on the whole, a rather beautiful theory to explain program behavior. There is but one basic flow. He "derives" the volume equation,  $V = N \log_2 \eta$ , as follows: Given  $\eta$  types,  $\log_2 \eta$  bits are required to uniquely identify them all. It requires  $N \log_2 \eta$  "mental discriminations" to select  $N$  symbols from a table of  $\eta$  symbols using a binary search technique since each search requires  $\log_2 \eta$  choices. Hence the volume.

This scheme seems to be based loosely on information theory, and this is where the measure fails from a theoretical viewpoint.

From information theory, we have the total information as  $I = -\sum N_i \log_2 P_i$  where  $P_i$  is the probability of the  $i^{\text{th}}$  type, and  $N_i$  is the number of times it is used. To get Halstead's volume, it would be necessary to assume that all types have equal probability, and that the types existent in a particular program constitute the universe of types, giving  $P_i = 1/\eta$ . Then  $I = -\sum N_i \log_2 1/\eta = \sum N_i \log_2 \eta = N \log_2 \eta$ . These two assumptions are untenable. Therefore, the Halstead measure fails to have a theoretical basis and must be thought of as completely heuristic.

Sullivan<sup>11</sup> reports a number of measures, some of which he rejects as either unworkable or not sufficiently studied, and others he proposes as measuring the quantities which contribute to complexity.

The  $C1$  complexity at any node of an elementary scheme is the number of paths containing more than  $X$  contiguous repetitions of a sequence of modes, where  $X=2$  unless otherwise stipulated. The complexity of the entire scheme is defined as the local complexity of the terminal node. A problem with this measure is that the measure can be very large or even infinite, the difficulty being in the notion of a loop as the immediate repetition of the same sequence of steps. If there are intermediate steps, this definition allows for the repetition of the same loop virtually an infinite number of times.

The  $C2$  complexity at any node is similar to the  $C1$  measure but eliminates the difficulty mentioned above. It is defined as one less than the number of paths from the start node to the given, not counting paths where any node occurs more than  $X$  times,  $X=2$  unless otherwise stipulated. The complexity of the entire elementary scheme is defined as the

local complexity at the terminal node. The complexity of a composite scheme is defined as the sum of the complexities of its elementary subschemes. Compared to  $C1$ , only the contiguity condition has been removed, but this removes the retracing of loops more than once for different sequences of paths. The subtraction of unity is to give unitary subschemes (sequences) a  $C2$  measure of 0 so that the sum of the measures for the elementary schemes of a scheme would not be sensitive to the incorporation of unitary decomposition or to decompositions along a single path.

The  $C2$  measure is always finite and if the program is structured, all of its elementary subschemes have measures of either 0 or 1. If the program is written using IF-THEN-ELSE for alternation and DO WHILE for iteration, then the  $C2$  measure is the number of IFs and WHILEs.

Sullivan feels that one weakness of the  $C2$  measure is that all paths are counted even when some paths do not contribute as much to complexity as others. He gives the example of an  $N$ -way case structure which is not necessarily  $N$  times as complex as a simple IF-THEN structure.

Measure  $C3$  at any node of an elementary scheme is defined as the first derivative of the  $C2$  measure with respect to  $X$ , the maximum no. of repetitions of any node. This eliminates the problem mentioned above with respect to the  $N$ -way CASE structure. The  $C3$  measure has not been studied in depth and Sullivan makes no further use of it.

Sullivan now uses another approach. The  $C$  measures are defined on properties of the control graph but completely ignore data. He defines two new values, the  $P$  measures, which describe complexity in terms of how the program interacts with the data.

Let the entire scheme be decomposed into subschemes and let the subschemes be modules (either actually or conceptually). Let the data subgraph proper to the module be that part of the data graph referenced from the nodes of the module or subscheme. If we now assume that any data element so referenced is relevant over any path through the subscheme, then it is reasonable to define measure  $P1$  as the product of the number of paths through the subscheme ( $C2 + 1$ ) and the number of data items (nodes of the subgraph) referenced.

As Sullivan himself states, the underlying assumption is not intuitively satisfying. Not all nodes of the subscheme reference every node of the data subgraph. Also, no distinction is made between set and use references.

Sullivan now changes perspective and examines each data item individually to see how it is used. For a given data graph nodes, define the process node set as all nodes which either set or use that data item, together with the start and terminal nodes. For nodes  $n_i, n_j$  of the set, let edge  $(n_i, n_j)$  exist if  $n_j$  can be reached from  $n_i$  without passing through another node,  $n_k$ , of the set. For this new control graph, the  $PD2$  data node complexity at this particular data node is the  $C2$  complexity of the control graph just defined except that any subscheme which has only use references and no set references in nodes other than its start node is assigned value 0. Sullivan has not determined how the  $P2$  measure should be defined from the  $PD2$  data node complexities. It may be

possible to have  $P2$  be the sum of all the  $PD2$  measures but he suggests that some other function makes more sense.

The various measures Sullivan suggests may be correlated with the number of bugs, or difficulty, of a program. At the time the paper was published, the measures had not been tested. This writer is not aware that experimental justification exists. The  $P2$  measure seems a bit difficult to implement. For each variable, it is necessary to create a control graph from the nodes of the program or system control graph which use that variable. Where there are subscripted or qualified variables, the problem is compounded greatly since the data graph will have many nodes, each of which requires the creation of a control graph. From a practical viewpoint, the calculation of this measure may require a program of considerable complexity to measure it.

## BACKGROUND ON INFORMATION THEORY

Suppose an alphabet of symbols is given  $(e_1, e_2, \dots, e_n)$  whose respective probabilities of occurrence are  $(p_1, p_2, \dots, p_n)$ . Then the following quantities can be defined or proven.

### *Uncertainty and information*

Where symbol  $e_i$  has probability  $p_i$ , the uncertainty associated with the occurrence of symbol  $e_i$  is  $-\log p_i$ . If the logarithm is base 2, the information is in bits. The uncertainty is a measure of the information provided by the symbol. Note that a symbol with low probability contains more information than one with high probability. Uncertainty and information are, for practical purposes synonymous in the present context.

### *Entropy*

The entropy,  $H$ , is defined  $H = -\sum p_i \log p_i$ . It is the average information provided by the alphabet of symbols  $(e_i)$ . We will discuss entropy again.

### *Total message information*

Let a measure be composed from an alphabet  $(e_i)$  whose frequencies of occurrence are  $(f_i)$  and probabilities are  $(p_i)$ . Since  $e_i$  contains  $-\log p_i$  bits of information, the total information contained in the message is

$$I = -\sum f_i \log p_i \text{ bits.}$$

This will form the basis of our complexity measure.

### *Ideal symbol length*

To minimize expected message length, where a message is defined as a string of symbols, it can be shown that the

encoded symbol should have a length in bits given by  $l_i = -\log p_i$ . For example, if  $e_1$  has probability  $p_1 = \frac{1}{2}$ , and if we use base 2, then the encoded length of symbol  $e_1$  should be  $l_1 = -\log_2 \frac{1}{2} = 2$  bits.

### Entropy, $H$

Entropy of an alphabet is defined as  $H = -\sum p_i \log p_i$ , summed over all  $i$ . If  $-\log p_i$  represents the length of symbol  $e_i$ , then the entropy represents the weighted average symbol length in an ideal coding scheme. The entropy will be used later in a complexity measure prediction formula.

### Message length

Let a message have symbol  $e_i$  occur with a frequency  $f_i$  and let an ideal coding scheme be used so that symbol  $e_i$  has coded length  $-\log p_i$ , then the coded length of the message will be  $L = -\sum f_i \log p_i$  where  $\sum f_i = N$  is the total number of symbols used in the message.

We see that the two calculations  $-\sum p_i \log p_i$  and  $-\sum f_i \log p_i$  can be interpreted in two ways. The first, entropy, is both the average information content supplied by the alphabet and the average length of the encoded alphabet where an ideal coding symbol is used. The second is both the total information contained in a message and the total message length if encoded with an ideal coding scheme.

These interpretations will be used again later when discussing the complexity measure.

### ASSUMED PROGRAMMING ENVIRONMENT

We require an installation at which statistics have been gathered on all items used in programs written at the installation. Specifically, the following are known: (1) the percentage of time each operator is used; for some operators several distinct percentages are known; these include operators which have nesting levels such as DO'S, IF'S, parentheses, and so on; (2) for operands such as variables, constants, arrays, function names, etc. we rank their probabilities according to frequency of use; that is, the probability of the most frequently used taken first, the next most frequent second, etc.; separate sets of probabilities are kept for arrays of differing dimensions and, perhaps, of different data types or structures; (3) frequencies for labels are kept as for variables, ranked in order of usage.

The required frequencies can be automatically accumulated as programs run at the installation. Different probabilities are maintained for each language for which a complexity measure is required. A number of programs have been written, which can be easily incorporated into a system, that count frequencies for FORTRAN, PL/I, and perhaps some others.

### A COMPLEXITY MEASURE

We now define a new complexity measure based on information theory.

Let a program be given. We count the frequency of occurrence of all tokens in the program: operands and operators. We assume that the long term probabilities of these are known and are represented by  $p_i$ . The complexity measure is defined as:

$$M = -\sum f_i \log p_i$$

There are several interpretations we can assign to this measure. First, from an information theory viewpoint, the measure represents the total information contained in the program, assuming the program to be a message.

Second, if we were to code each token of the program using an ideal coding scheme, the measure would be the total length required to encode the program.

The measure as defined is sensitive to the frequency of usage of all symbols and to the proportion of times the symbol has been used in the past.

The  $p_i$  are an accumulated statistic gathered at the installation over a period of time. It is quite possible that a given program may have a different measure at different installations. This would reflect different programming techniques at these installations. There are both positive and negative aspects to such a situation.

On the positive side, if a program manager has a known correlation between this complexity measure and programming difficulty at his installation, he can predict how long it will take his programmers to finish the job. This would be of primary importance to him.

On the negative side, the manager has no way of knowing from the measure whether his programmers are working efficiently. If it is important to know this, and one would think it is, a comparison of the measure would have to be made with another installation's. If another installation has a different set of probabilities, the measure can be recalculated using those probabilities. It is somewhat unclear, however, whether the results are meaningful.

A better way to measure programmer efficiency might be with the entropy measure to be defined later.

This measure may appear, to some, to be similar to Halstead's program length measure. There is, however, a major difference. Halstead's "probabilities" are calculated as  $f_i/n$  for a given program. The measure here uses long term probabilities obtained over a period of time. Halstead makes no distinction between programs using difficult constructs and those using simpler ones. The measure proposed here does.

### ADVANTAGES

The measure, as defined, has the following advantages: (1) once the programming has been done, calculation of the measure can be completely automated; (2) the measure can

be estimated at an early stage to detect programs or modules which may be overly complex; (3) the measure is more sensitive to infrequently used tokens than more frequently used ones; this is intuitively satisfying as it is reasonable to suppose that people will tend to make more errors in using things with which they are less familiar; (4) most importantly, the measure makes distinctions between tokens which are used at different levels; the probability of a DO used as a second level is lower than the one used at the first level; the level two DO will contribute more to the measure than the level one; this also seems quite satisfying as it is reasonable to suppose that nesting is more complex than sequencing; very few of the measures proposed so far have considered this aspect of programming.

## ENTROPY

We can define a language entropy as:

$$H = - \sum p_i \log p_i$$

A language with a high entropy has many tokens with low probabilities. One with low entropy has few tokens with high probabilities.

Note that a given language, such as PL/I, will have different entropies at different installations. At one place, programmers might write programs containing a large number of variables with resulting low frequencies. The other installation might use a great deal of segmentation with the result that each module has only a few variable of high probability. The second installation would tend to write programs with lower complexity than the first.

The entropy might, therefore, be a measure of an installation's ability to reduce complexity through segmentation. It could also, perhaps, be used to compare with other installations. For example, if one installation used the various language features more uniformly than another, then the entropy would be larger at the installation. This might indicate a better knowledge of the language, perhaps.

## ESTIMATING THE COMPLEXITY MEASURE

We have seen that the entropy,  $H$ , is the average length of a program token and the complexity measure,  $M$ , is the encoded length of the entire program.

Suppose  $\bar{N}$  is the expected number of tokens, operators or operands, in a program. Then we can define an estimated complexity measure,  $\bar{M}$ , as

$$\bar{M} = \bar{N}H$$

The only problem is obtaining  $\bar{N}$ . Some work is being done along these lines. For example, Professor Laemmel,<sup>12</sup> at PINY, has been using a modified Zipf's law to obtain the number of tokens if the number of types is known. The number of types might be estimated at an early program state. Halstead, using software physics, attempts to estimate the number of tokens, which he calls the length, from the number of I-O parameters.

Whichever scheme is used, an estimate can be obtained.

## CONCLUSION

A measure has been proposed which, it is to be hoped, correlates well with difficulty in programming. The measure has mathematical foundation in information theory and is intuitively satisfying as well. It can be easily found after some initial programs have been added to a system. Furthermore, a predictor for the measure is available.

The measure is based on the probability with which various tokens of a program are used. The probabilities are collected over a period of time and are cumulative.

The measure is installation dependent but comparisons between installations is possible by use of an accompanying measure. This other measure is also used in predicting to complexity measure.

## BIBLIOGRAPHY

1. Hellerman, L., "A Measure of Computational Work," *Transactions on Computers*, Vol. C-21, May 1972, pp. 439-446.
2. McCabe, J., "A Complexity Measure," *Transactions on Software Engineering*, Vol. SE-2, Dec. 1976, pp. 308-320.
3. Mills, H. D., "Mathematical Foundations of Structured Programs," Federal Systems Division, IBM Corp., Gaithersburg, Md, FSC72-6012, 1972.
4. Meyers, G. J., "An Extension to the Cyclomatic Measure of Program Complexity," *SIGPLAN Notices*, Oct. 1977, pp. 61-64.
5. McTap, J. L., "The Complexity of an Individual Program," published in these *Proceedings*.
6. Chapin, N., "A Measure of Software Complexity," *Proceedings of the 1977 NCC*, Montvale, NJ, AFIPS, 1979, pp. 995-1002.
7. Chen, E. T., "Program Complexity and Programmer Productivity," *Transactions on Software Engineering*, Vol. WE-4, May 1978, pp. 187-194.
8. Mohanty, S. N., "Models and Measurements for Quality Assessment of Software," *Computing Surveys*, Vol. 11, No. 3, Sept. 1979, pp. 251-275.
9. Schutts, D., "On a Hypergraph Oriented Measure for Applied Computer Science," *Proceedings of COMPCON*, 1977, pp. 295-296.
10. Halstead, M. H., *Elements of Software Science*, Elsevier North-Holland, Inc., New York, 1977.
11. Sullivan, J. E., "Measuring the Complexity of Computer Software," MITRE Technical Reports, MTR 2648V, June 1973.
12. Shooman, M. and Laemmel, A., "Statistical Theory of Computer Programs in Information Content and Complexity," *Proceedings of COMPCON*, 1977, pp. 341-347.





## **Software Engineering Education**

Several studies have indicated that the data processing industry's most critical problem during the 1980's will be a shortage of qualified software engineers. This panel brings together a number of people who have been actively addressing the problem of increasing the supply of qualified software engineers through both University and industry programs in software engineering education. They will address the following issues:

- What skills will be needed by the software engineer of the future?
- What software engineering programs are currently underway in universities, industry, and professional societies to meet these needs?
- What particular approaches have been tried to date, and how well have they worked out?
- What are some resulting guidelines for mounting a successful software engineering education program?



**Barry Boehm**  
*Area Director*



### **Special Topics**

In this session, an integral EDP auditor discusses his function and involvement with data processing and the users of data processing. He shares his views on a wide range of specific EDP areas of activity and involvement. He will relate his concerns to control objectives in the audit methodology utilized for determining that the control objectives are effectively assured.



Gene Smith  
*Area Director*



# Technology development, severed ventures, and other aspects of corporate venture capital

by JEAN E. DE VALPINE

*Memorial Drive Trust*  
Cambridge, Massachusetts

## INTRODUCTION

The following discussion is based on the experience of Memorial Drive Trust (MDT) in connection with efforts of business corporations to participate through investment or contribution of technology, resources, or personnel, in the development of other business enterprises. We shall emphasize so-called "severed ventures," but will include an example of another approach. We shall also describe a capability in the areas of technology and product development up to and including participation in startup or early-stage technology ventures.

MDT, in accordance with its charter, views itself as being in the business of investing in and contributing on the directorate level to the maturation of enterprises comprising a very wide spectrum of business categories.

We were founded in 1951. During our first fifteen years, aside from our investment in Arthur D. Little, Inc. (ADL), we were predominantly engaged in real estate activities. Real estate development and ownership surely constitute a type of venture-capital activity within a broad definition of the term. That, however, is not the subject of principal interest to this audience.

During the early sixties we began to shift emphasis into the area of venture-capital investment in small prepublic enterprises, with a bias toward "high technology" content, and we have stayed on this course ever since. It is no secret, however, that MDT is widely perceived in association with ADL. It is fair to say that this perception has some validity in view of the fact that up until 1969 MDT owned 100 percent of the stock of ADL, and after 1969 and to the present has owned approximately 70 percent of the stock of ADL.

This association has given a corporate venture aspect to some of our activities. Nevertheless, management and operations of MDT are separate and distinct from management and operations of ADL. In general, we have operated and can operate in either the corporate venture mode or the independent venture-capital mode.

## A CAPABILITY IN TECHNOLOGY AND PRODUCT DEVELOPMENT

In fact ADL itself has an active venture-capital activity called Arthur D. Little Enterprises (ADL Enterprises). In

some respects, though not in all, it seems to me that ADL Enterprises has similarities to Scientific Advances Inc., the venture-capital subsidiary of Batelle Memorial Institute. ADL Enterprises includes in its domain managing and developing to feasibility or commercialization of inventions, including the handling of patent strategy. It also seeks possible spinoffs of technology and/or viable small businesses from within ADL, somewhat after the fashion of General Electric's Technical Ventures Operation. ADL Enterprises further seeks equity in other companies, in situations in which ADL can contribute technology or expertise in exchange for equity or options on equity, or perhaps alternatively on a royalty or other licensing basis. In appropriate circumstances, ADL Enterprises may also make cash capital investment.

ADL Enterprises constitutes an evolution of ADL's Invention Management Department, formed in 1958 to evaluate, process and facilitate the exploitation of proprietary inventions developed by ADL staff members, as well as those submitted from outside the company. The scope of this activity now includes active involvement and participation in new product ventures.

For inventions submitted from outside, ADL Enterprises stresses those that represent substantial technical achievements and not minor improvements, design modification, or styling changes. They are not ordinarily interested in household devices, automotive accessories, toys, games, and wearing apparel. Although ADL can act strictly as a licensing agent for fully developed inventions, it customarily manages technology to which engineering, economic, or marketing value can be added in order to maximize licensing potential. A new product invention should represent a large nongovernment market with sales potential of at least \$5 million annually. A manufacturing process improvement should permit yearly savings of at least \$250 thousand annually.

If after evaluation tests ADL Enterprises should decide to pursue exploitation of the invention, they will propose an agreement to create a profitable licensing situation. This usually will involve a grant to ADL of exclusive rights for a limited time to develop and license the invention, and will normally include technical and commercial assistance in order to enhance the technology. ADL will normally assume patent costs after an agreement has been made. Typical

agreements share licensing income on a 50/50 basis, but unusual circumstances may warrant a different formula.

Based on success in development and licensing of new technology, ADL Enterprises has broadened its objectives to include: a) exploration of totally new market segments for a product already fully commercialized in a different field; b) provision of assistance to small firms lacking the necessary managerial-technical talents or finances to capitalize on valuable technology; c) cost-sharing in the development of new technology; and d) joint-venturing arrangements, including equity arrangements alluded to above. By virtue of its depth and breadth of resources, ADL is able to draw from a large pool of professionals and specialists in the major disciplines to perfect and commercialize new technology. They can then combine their knowledge of industry patterns in the marketplace with an ability to reach appropriate decision-making individuals to consummate optimum new product venture, licensing or market agreements.

Again, it should be emphasized that ADL Enterprises, like ADL, constitutes an activity organized and operated on a basis of total independence from the investment and new-venture operations of the parent, MDT. Transactions or activities in which the participations and contributions of MDT and ADL Enterprises may be either complementary or compatible are negotiated and structured strictly on an arm's length basis.

#### CERTAIN INVESTMENT DESIDERATA

With respect to MDT a brief comment is in order as to what we view as ideal venture-capital criteria, to be followed by some observations based on our experience in the corporate venture mode.

I do not think it useful to try to be too precise in describing what we seek in the way of investment opportunity. Suffice it to say that the range is \$250 thousand to \$2 million. We do some startups, but relatively few as a percentage. We look for evidence of outstanding potential for growth in corporate level management skills, technology, marketing expertise, and manufacturing capability. We have been and are majority shareholder in a number of companies. However we do not seek this role. Rather, in general we prefer to be one of several professional venture-capital investors.

Ideally we like situations in which ADL technical or marketing expertise can assist us in technology assessment and evaluation of market potential. However, just as the involvement of Scientific Advances Inc. does not carry with it access to free Batelle R&D, likewise our involvement does not carry with it access to free ADL professional services.

On the other hand, our familiarity with the capabilities of various ADL sections and divisions, and with the strengths and interests of outstanding individuals, is I believe rightly perceived as an asset which enables us to make highly efficient and effective use of a wide range of skills and knowledge. It is this characteristic which results in other venture capitalists inviting us to participate in their deals, either to strengthen their own evaluation process, or potentially to aid the enterprise subsequent to investment.

#### SEVERED VENTURES

I now turn to the subject of so-called "severed ventures." In the following discussion we use the term "sponsor" or "sponsor company" to refer to the operating business enterprise which may contemplate, adopt, or execute, either as a policy or randomly, the identification, preparation, and "spinout," "spinoff," or "severance" of activities developed by and theretofore conducted within the sponsor company. Needless to say the terms "spinoff," "spinout" and "severance" are here used synonymously. In general the term "severed venture" will be used as the generic term.

Of course the same company may also contemplate, adopt, or randomly execute a policy of venture-capital investment in small pre-existing outside enterprises, either by means of acquisition of equity or quasi-equity for cash, business assets, or knowhow or technology transfer. This, however, constitutes another and distinct mode of corporate venturing.

Corporate strategic planners can and have devised many modes and motivations for sponsor company involvement in severed ventures, such severed venture companies being constituted to operate autonomously with respect to the sponsor company, and to engage in technology related or unrelated to, or only peripherally or potentially related to, areas of interest within the mainstream of the sponsor's business or business plan. In order to provide a concrete framework for purposes of observations from our experience, I shall give some examples. These examples, although adhering rather closely to fact, will be rather thinly disguised by means of nonmaterial permutations of names and facts.

#### *Cryotec Inc.*

Cryotec Inc. was severed from ADL about ten years ago. At that time revenues were in the \$4-5 million range. In 1979 revenues are approaching the \$80 million level. The cryogenics hardware business of ADL had evolved and been stimulated during the period when ADL was owned by MIT. The business grew slowly during the period following the acquisition by MDT from MIT of the common stock of ADL.

As a division of ADL, the cryogenics activity was at best marginally profitable. Indeed it is fair to say that the business was so embedded in the ADL R&D environment that it was extremely difficult to determine what might be its potential profitability as a discrete activity. This is a problem endemic to potential technology severances from any sponsor company having a high level of internal R&D. Suffice it to say that in the ADL environment the business was conducted in the ADL tradition of research, development and custom engineering of the highest quality. Although not demonstrably incrementally profitable to ADL from an accounting standpoint, such high-quality input provided the business with an outstanding reputation which proved to be of great value during the years of transition to genuine stand-alone independence.

The question of motivation is a basic conceptual issue for any sponsor company contemplating a program, policy or

random effort to identify and launch, as an independent severed venture, some discrete segment of its own business. What are to be the reasons and criteria for severing out a given segment of business? In the case of the severance of Cryotec from ADL the rationale was that a production-oriented activity was not in the mainstream of the traditional consulting and R&D nature of ADL's business and image, that the cryogenics business would indeed benefit by removal from the hothouse R&D atmosphere of ADL and exposure to competitive demands of the commercial marketplace, and further that, so far as could be ascertained, ADL could achieve at least as high return on equivalent investment redirected in traditional channels.

Initially the stock of the severed venture, Cryotec, was held roughly 80 percent by MDT and 20 percent by employees. ADL retained no continuing role of guidance or overview. At the present time (ten years later), as a result of several acquisitions MDT holds about 54 percent of the stock, the balance being in the hands of the public, including employees.

The business was completely severed physically from ADL and moved to a separate location. The board of the newly-severed venture consisted of two representatives of MDT, including myself, and three members of management, all of whom were former ADL staff members. All this illustrates a number of difficult questions which arise when one contemplates modes and techniques for creating an independent enterprise out of segments of an existing enterprise. Rather than list modes and discuss pros and cons, let me state some tentative conclusions.

For one thing, ultimate realization of value tends to be optimized if the severed venture can be accomplished so that it does not require over the period of the first several years any third party cash investment for working capital or other purposes. This is an ideal which happened to have been realized in the case of Cryotec. However, I would guess that generally speaking the motivation of the sponsor company would include the objective of reallocating working capital or other resources from the severed venture to ongoing or proposed operations of the sponsor. This would usually require third party investment at the outset.

Further, the management of the severed venture may view the sponsor as remaining ambivalent with respect to commitment to follow through to an ultimate achievement of permanent independence of the severed venture if outside investors are not introduced. Indeed it is natural that sponsor companies contemplating a well-thought-out severed venture program tend to be ambivalent in this respect, resulting at least from the outsider viewpoint in inconsistent and in some cases contradictory policy or practice. This is evidenced in situations in which the sponsor company attempts to negotiate the equivalent of the right to reacquire and reabsorb the severed venture within a given period of time, or contingent upon certain events.

Another important element in our view is that the organization within or affiliated with the sponsor company, and having the function of identifying the activity to be severed, executing such severance, and exercising the continuing stockholder interest of the sponsor with respect to the sev-

ered venture, should be autonomous within the management structure of the sponsor, should be professionally competent and dedicated to venture capital management, and should have continuity, coupled with a long-term charter. If this independence is not present a likely result is the inhibition and hobbling of the severed venture by reason of subjection of its decision-making processes to the ordinary procedures of the operating management hierarchy of the sponsor.

A related issue might be termed the "identification problem." Who identifies a candidate? What are the criteria for such identification? Where does the final decision lie? All this relates to motivation and the definition of objectives with respect to a severed venture program. I submit that within any given sponsor company, and especially a very large sponsor company, the set of skills required in connection with the identification problem may be somewhat different from the set of skills ordinarily associated with a private, independent venture-capital investment enterprise.

Of course the identification problem may be either more or less of a problem depending on policy criteria established by the sponsor company. Such criteria might include the following: only viable businesses of a certain magnitude, say \$1 million sales or more, having a pro forma record of profitability and/or positive cash flow, and having a high probability of surviving as severed ventures without any assistance whatever from the sponsor company. Of course these need not be the criteria. One could adopt a policy of spinning out startup technology looking for a market. My personal belief, however, is that a serious, well-thought-out, long-term severed venture program to be successful must rest on a firm policy of spinning out only such sponsor company activities as can be identified and segregated in a manner such that the severed venture has a visible likelihood of viability over the first several years of independent existence.

Interesting questions arise in connection with the negotiation of details with respect to the severance from the sponsor company of the to-be independent severed venture. The negotiators include at least the prospective management of the severed venture, some of whom may be recruited from the outside for the purpose at hand, the management or division management of the sponsor company, the management of the autonomous organization or subsidiary within the sponsor company having purview of the severed venture program, and in most cases prospective outside cash investors. I might insert parenthetically that we at MDT have been in the shoes of both prospective outside cash investors and the severed venture program management organization, except that in negotiating with the sponsor company we have had the unusual advantage of being the controlling stockholder of the sponsor. Even so, the negotiations viewed from the standpoint of the severed venture program management entity can be difficult and delicate. For example, in the case of Cryotec a serious issue arose as to the extent to which ADL would retain a cryogenics capability, potentially capable of competition with the severed venture. We know this to be one of the most common issues in connection with other severed venture activities of which we are aware. It turned out that ADL did retain, on a very advanced level,



an R&D capability in cryogenics. However no competitive situations have arisen, except perhaps with respect to certain NASA and DOD programs.

The lesson, however, is that the sponsor must carefully think through and contractually delineate the ongoing relationship of the sponsor to the severed venture with respect to patents, technology, and competition. These matters will depend upon a clear understanding by the sponsor of the sponsor's ultimate intent and objectives with respect to the severed venture. Here again the sponsor confronts the possibility and perhaps even the likelihood of incompatible or contradictory objectives or practices. The objectives may even be subject to change, and given the nature of the business organism changes of objectives may be probable. A simple example is the case of Cryotec. The original mandate to me from the MDT Board was to do what could be done to validate Cryotec as a small advanced technology activity with pretax return on investment on the order of 20 percent, and having achieved that goal, to sell or merge the severed venture into a larger company.

However, when that goal was achieved Cryotec then appeared to be justified on its merits as a continuing investment, and consequently the original objective was not executed. In some situations, but not necessarily all, such a change of objective might be disappointing to the severed venture management who might have hoped for an eventual public offering which would remove absolute majority stock control from the hands of a single entity, or who might have hoped for a merger into another entity out of which the severed venture management might emerge dominant, due to the relatively equal sizes of the merging companies.

A further question arising at the outset of the severance process is who should be CEO of the severed venture. Should the CEO come from inside the sponsor company or should he be recruited from the outside? Whichever course is followed a certain amount of luck is involved.

In the case of Cryotec the MDT Board advised that I should recruit a CEO from the outside. After interviewing all of the divisional management personnel of the cryogenics operation as it was constituted within ADL, I selected the individual who had the function of production management of the business. He turned out to be, and is widely recognized as, an outstanding CEO. I had no elaborate rationale for the choice, except that I was confident as to market acceptance and dominance, and was concerned mainly with the capability to produce profitably and cost effectively once the activity was moved out of ADL R&D environment.

#### *Hyperballistics, Inc.*

Our experience with Hyperballistics Inc. would make a fascinating business school case. Strictly speaking this was not an ADL severed venture. It was instead the severance of a joint-venture controlled and managed by ADL and having a third party stockholder who had acquired stock for cash. This third party stockholder quickly took the position that it had made its cash investment on the understanding

that ADL would be responsible for the management of the enterprise. The third party stockholder made no distinction between ADL and MDT, and when MDT succeeded to the ADL stockholder position the third party stockholder was instantly on the phone demanding that I become Chairman of the Board of Hyperballistics.

This illustrates still another problem for a would-be sponsor company. The problem is that third party investors, with or without justification, tend to place significant weight on the fact that the severed venture is coming out of a highly-reputable sponsor company. They tend to believe that somehow, although the intention may have been disclaimed, the sponsor company will oversee and guide the management of the severed venture, and will in dire distress support and come to the aid of that venture.

For us at least, Hyperballistics also taught the lesson that a severed venture, or for that matter a corporate venture-capital involvement of any kind in an independently constituted enterprise, should be premised on that enterprise having its own full complement of managerial capability, and not being dependent by contract or otherwise for operational level management provided from within the sponsor company.

#### *Technical Ventures Operation*

We have had the role of cash investor in certain spinoffs executed by the former Technical Ventures Operation of General Electric Company. In each case these severed ventures have survived over a period of several years, and appear to have established viability as small businesses.

In each case the severed venture was constituted with adequate revenues and resources reasonably to assure the achievement of modest positive cash flow and profitability.

At the time of these spinoffs the Technical Ventures Operation adhered to the "three-legged stool" policy. This meant that none of the three parties in interest, namely GE, the management, and ourselves—the cash investor, should hold an absolute majority of the voting stock. Each of the three parties was represented on the board of directors with GE providing two board members, one representing the Technical Ventures Operation, and one to provide oversight and guidance to the severed venture by reason of special qualifications including expertise and experience in the technology or markets.

On the whole I think there is merit in this paradigm. I know that management does in fact view the "balance of power" effectuated via the three-legged stool mechanism to be very important. I believe that most cash investors would deem this attribute important. On the other hand the positive psychology of this arrangement tends to be negated to the extent that the sponsor attempts to retain rights to reacquire the severed venture. Further, the cash investor should be confident of his capability to make independent technical and market evaluations of proposed spinouts from other corporations.

## ANOTHER STRATEGY—INFILTRATE, WATCH, ACQUIRE

It is my understanding that the Chairman of EG&G has described their policy for developing new ventures as utilizing a number of different mechanisms including "minority positions" without buyout agreements. They recognize that it is usually not desirable to be locked into a minority investment with no public market, but have decided to rely on persuasion rather than contract for ultimate merger.

An interesting variant of this strategy has, in fact, been successfully executed by Time Incorporated in the acquisition of American Television & Communications Corporation (ATC).

By way of preface it should be recalled that MDT during the sixties acquired a group of cable television systems of the classic variety, and placed these systems in a wholly-owned operating subsidiary, Oregon CATV, Inc. In the late sixties Oregon CATV, Inc. was combined with other aggregates of systems including the systems of Narragansett Capital Corporation and Boston Capital Corporation to form ATC. A public offering followed.

Over the following decade ATC grew rapidly while maintaining extremely strong financial condition. As a result ATC was widely perceived by industry analysts as the premier company in the industry. The market generally reflected this appreciation by awarding ATC shares a premium price in relation to the industry as a whole. Throughout this period I served on the Executive Committee and Audit Committee of ATC.

During the early seventies various analysts and writers in business publications delivered themselves of the opinion that Time Inc. had at least three major problems, including *Life Magazine* and Time's cable television operations and franchises. Within a rather brief interval, Time, Inc. ceased publication of *Life* and merged its cable television systems into ATC in exchange for ATC common stock on the order of 10 percent of the outstanding common stock of ATC. At the request and encouragement of the ATC Board, the President of Time Inc. joined the ATC Board.

ATC continued to flourish. An outstanding set of capabilities in system management and franchise acquisition was structured, and maintained, and improved. The systems and franchises contributed by Time Inc. were operated and developed with correspondingly excellent results.

In consequence, in 1978 Time Inc. made known their intention to acquire additional shares of ATC on the market and from various institutional holders on the basis that the ATC stock as such was viewed by Time Inc. as a good investment.

Subsequently, after having built their stockholdings to a position in excess of 20 percent of ATC outstanding stock, Time Inc. expressed to the ATC Board a desire to open negotiations with a view to acquisition of ATC in exchange for stock and cash. In late 1978 this acquisition was consum-

mated on a basis attributing an implicit value to ATC on the order of \$250 million.

It is true that Time Inc. paid a full and fair price for ATC. It is also true that they were not in a position to compel the ATC Board or stockholders to agree to merger. Nevertheless their familiarity with ATC, their presence on the Board, their appreciation of the turnaround accomplished by ATC in the operation of their former systems, and ultimately and especially their significant stock position, placed them in a very favorable position to observe, and to exercise the options of either holding ATC stock as an investment, selling that stock, or attempting to acquire ATC. The presence of Time Inc. as the largest stockholder of ATC tended to inhibit other potential acquirers.

## CONCLUSION

If success be defined in terms of stand-alone viability as a small business, experience has shown that business enterprises can indeed be successfully identified within and severed from larger business entities. In our experience such successful severed ventures have constituted viable business units existing within the sponsor corporation prior to severance.

It would seem to be inadvisable to attempt to set up as a severed venture a startup technology, or a technology or product line not having found a meaningful market niche. With respect to such technology or product line the would-be sponsor should either undertake internally, or in collaboration with an external partner skilled in new technology and product development, the maturation of the technology or product line to a level of feasibility and commercialization, prior to attempting a severance.

It is essential that a severed venture program should have very careful thought-out objectives, definition of candidates for possible severance, procedures for identifying such candidates, severance decision processes, and structural paradigms for severed ventures. It is equally important that the execution of such program should be delegated to a professionally competent organization or subsidiary, reporting at a high level of the sponsor corporation, vested with sufficient independence and autonomy, and having a time horizon on the order of ten years.

In the execution of its mandate, such venture-capital arm of the sponsor corporation should acquire and hone the skills necessary to identify possible candidates for severance, and to negotiate in detail with parties in interest, including the sponsor corporation, with respect to the terms of severance, including ongoing future relations relating to patents, technology, competition and the basis, if any, for the provision by the sponsor corporation of guidance, assistance, and technology. And, of course, such venture-capital arm should possess the capability to provide continuing board level guidance and surveillance to severed ventures in its portfolio.



# Recommendations for increasing the availability of capital\*

by RICHARD C. PFLAGER

Control Data Capital Corporation  
Minneapolis, Minnesota

## INTRODUCTION

Mark Twain's proverbial statement about the weather—"Everybody talks about the weather but nobody does anything about it"—might be equally applicable to small businesses. Everyone seems to be talking about small businesses\*\* these days, but is there really enough that is being done to help them? There are, to be sure, a number of organizations that work on their behalf such as the Small Business Administration in the government sector and venture capital firms in the private sector. There has also been, in recent years, a rapid increase in the number of small business investment companies (SBIC's) as well as minority enterprise small business investment companies (MESBIC's) both federally licensed under the Small Business Investment Act of 1958. More encouraging than the increase in the number of these organizations is the increase in investment funds that they have made available to small businesses: \$33 million for MESBIC's and \$255 million for SBIC's as of March 31, 1979.

While all of this is very good and while it can be argued that it is extremely encouraging, I would ask again, is it sufficient? Small businesses are, after all, a major source of jobs within the U.S., outpacing the major corporations with impressive growth in the rate of employment. Small businesses have also been the "well spring" of many new and improved products, services and technologies that contribute significantly to our economy's productivity, the only successful damper to inflation that we know. Together with agricultural commodities, these technologically intensive products comprise the heart of our export trade.

The purpose of this paper is to recommend certain changes to our existing tax laws, and by so doing, increase the availability of capital for the small, technically oriented companies that have served us so well in the past as the mainstays of our economy.

\* This paper is derived from a more comprehensive report to the Assistant Secretary of Commerce for Science and Technology prepared under the direction of William C. Norris, Chairman of the Board and Chief Executive Officer of Control Data Corporation.

\*\* Throughout this report small businesses are defined as those that have less than 500 employees, are not majority owned by larger firms, are operated for profit, and are involved in the creation and use of new knowledge, products, processes, or services. Activities related primarily to real estate transactions are excluded.

## RECOMMENDATIONS FOR INCREASING THE AVAILABILITY OF CAPITAL

### *Capital gains taxation*

The present level of capital gains taxation has become a very critical constraint on the founding and expansion of small, technically oriented firms. Increases in capital gains taxation are probably more responsible than any other factor for the gradual deterioration in technological entrepreneurship that has occurred in the United States during the last decade. Such changes have successively lowered after-tax returns for investors in successful innovation to a level where now, technologically innovative firms no longer are able to attract adequate investment.

Engaging in industrial innovation has always been inherently risky because the uncertainties associated with new technology developments are always compounded by the uncertainties of market acceptance of the new products and processes that result from such developments. At the same time, innovation is usually a capital intensive activity, not so much because it requires a massive investment as do steel and chemicals, but because of the extensive time lag between the launching of the development and the establishment of its large scale acceptance in the marketplace. During this time, capital is required to cover the expenditures for start-up costs before the revenues have begun to be realized. Such capital is forthcoming only when potential investors believe that the after-tax returns will be adequate to cover their risks. The problem of adequate rewards, however, is not just one for capital. Traditionally, key management and technical personnel have been compensated for the personal risks in joining uncertain ventures by sharing in the fortunes of the firm rather than by receiving salary payments. In our free enterprise system successful entrepreneurship creates the economic values. These, in turn, are reflected in the rise in stock prices of the enterprise and are realized by investors and key individuals in the sale of their stock in such enterprises. Thus the after-tax capital gain is of critical importance if we are to have innovation by small firms.

In looking back over the last decade, the tax on capital gains from 1969 to 1977 increased dramatically. Prior to 1969, the maximum capital gains tax rate paid by individuals was 25 percent. The Tax Reform Act of 1969 increased that rate

to a maximum of 40 percent—a 35 percent rate on the capital gains and an additional 5 percent from the operation of the minimum tax. Legislation also reduced the tax on earned income from a maximum rate of 70 percent to 50 percent. Thus the differential between the taxation of salaries and capital gains narrowed from 70 percent on salaries and 25 percent on capital gains to 50 percent and 40 percent respectively.†

The Tax Reform Act of 1976 provided for further increases in the minimum tax as well as raising the maximum rate on capital gains to approximately 49.0 percent. These changes virtually eliminated the differential between the rates on earned income and capital gains that existed prior to 1969. The effect of these changes was further compounded by the high rate of inflation which produced significant capital gains in current dollars, and hence capital gains taxes, for assets whose value after adjustment for inflation had actually declined. The impact of such changes in taxation has been dramatic for the small technically oriented firms in which the prospect of capital gains has been the major incentive for investors. The 95th Congress recognized the negative consequences of the high rate of capital gains tax by passing significant rate reductions. The legislation did not, however, restore the rates to the 1969 level. Given the risks of small, technically oriented businesses, a further rollback is necessary for these firms to realize their growth potential in such vital areas as job creation. It is also necessary to consider an even lower rate of 10 percent to attract investment in the smallest of businesses; for example, application of the lower rate should be determined by the size of the businesses at the time the investment is made and thus serve to attract capital to new firms and to recognize the higher degree of risk in the smallest firms.

Therefore, our highest priority is for a capital gains tax reduction that is targeted for small, technically oriented firms. Such a tax reduction would be a superior method of improving the availability of capital. By increasing the rewards for successful ventures, an incentive could be provided to manage such enterprises in an efficient way, leaving to the marketplace the distribution of these incentives among the various firms. This approach would be superior to providing loans, or other federal financing to small firms; approaches that would thrust upon the federal government the difficult task of deciding among the different loan applicants. This proposal might result, at least initially, in revenue loss to the federal government, but given the narrowly limited target of the proposed tax reduction, it would be a minimal one, and losses would be offset by the gains in employment and output from the successful firms.‡

#### **Recommendation 1**

That the capital gains tax rate be reduced to 25 percent (the pre-1969 rate) on the capital gains realized from the sale

† *Tax Policy, Investment, and Economic Growth* (A report by Securities Industry Association, 1978), p. 63.

‡ *Tax Policy, Investment, and Economic Growth* (A Report by the Securities Industry Association, 1978), pp. 34-7.

of stocks of small businesses whenever such stocks have been held for three years or more, with a rate of 10 percent for the capital gains of investors in the smallest of businesses. This reduced rate would not be applicable to any capital gains realized from real estate sales.

#### *Tax-free exchange of stock*

Continuous investment holdings are risky even in small, technically oriented firms whose stock has risen in value. The reason being that stockholders have a propensity to diversify their investments. Under existing tax laws the most profitable way to diversify is through a tax free reorganization with a large firm carried out through a tax-free exchange/transfer of stock. Investors oftentimes find that equity shares of large firms are likely to be more liquid and represent a diversified set of economic activities. On the other hand, this method of diversification tends to concentrate capital in the larger firms.

It is important, therefore, to have tax policies that encourage the continuous use of capital in the start-up of new firms. At the same time the investor's desire for diversification of risk is a legitimate one and must be recognized. Accordingly there is a need to establish an alternate route for tax-free diversification of risk that would encourage the formation and growth of small firms but allow the tax free roll-over of investment from one small firm to another. Such a provision—similar to the roll-over provision on sale of homes—would make funds available to new, small, technically oriented firms, from the most knowledgeable and receptive of investors—those that have already participated in such ventures. It would remove, moreover, the tax incentive for the sale of the successful small firms to the large ones, thus preserving the small firms as independent business entities. It would also allow the investor to diversify his holding in several small, technically oriented firms.

Essentially this same proposal was made in 1976 by the Tax Policy Task Force of the Small Business Advisory Committee on Economic Policy.

#### **Recommendation 2**

That appropriate changes be made in the tax code to permit deferral of capital gains taxes on the sales of shares in small businesses if the proceeds are reinvested within one year in one or more other small, technically oriented firms.

#### *Taxation of corporate income and tax treatment of start-up losses*

##### Taxation of corporate income

Small businesses frequently experience great difficulty in obtaining capital not only in their early, formative years, but also during the years of their rapid expansion. Firm data are not readily available, but capital shortages during this period

are believed to contribute greatly to the high failure rates of small businesses. Causes of capital shortages cover a broad spectrum, but in the case of the small, struggling companies that bring new products or services to the market, current tax rates on net earnings are so high as to preclude the establishment of a solid, financial base that is attractive to investors. The best and easiest way for small firms to achieve a sound financial basis, and hence adequate funds to support expansion, is, of course, through retained earnings. Current tax rates on corporate earnings are not, however, sufficiently differentiated between the small firms and the large, more established corporations. Net earnings of all domestic companies (other than mutual savings banks, life insurance companies or regulated investment companies), regardless of size and age, are subject to a tax of 17 percent on the first \$25,000 of net income, 20 percent on the next \$25,000, 30 percent on the next \$25,000, 40 percent on the next \$25,000 and 46 percent of that portion of the taxable income that exceeds \$100,000. And yet, the tax bite doesn't end there. Most states also collect income tax on small businesses, and many impose taxes on dividends to stockholders.\* Small businesses would have a better chance for survival, as well as growth, if the tax rates on net earnings were also reduced.

#### Start-up losses

The well established corporation is also provided a tax incentive for innovation insofar as its expenses for the early phases of innovation are a deduction from its corporate income tax. The new, small firm cannot obtain this tax benefit since it lacks the profits from which such losses can be deducted. Then too, such losses (incurred after December 31, 1975) can be carried forward seven years and charged against income. Before 1976, net operating losses could be carried forward for only five years. It is common knowledge that some of the most advanced and promising technology has a longer gestation period than seven years and hence does not yield profits within a seven-year period in which to take advantages of earlier losses through offsets. In short, there is a tax bias against the smaller firm that is developing technology when compared to the larger firm. This unfavorable tax bias should be eliminated.

#### **Recommendation 3**

That the threshold for the application of the full corporate tax rate of 46 percent be raised from \$100,000 to \$250,000 of annual net income; and for annual net income below \$250,000, a progressive rate schedule be established beginning at 10 percent on the first \$20,000, and increasing in 10 percent increments to a ceiling of \$250,000 on each additional \$40,000 until \$100,000 is reached and then no increase until \$250,000; in addition, the carry-forward provisions for start-up losses of small businesses be extended from seven to ten years.

\* *Tax Review*, Vol. XXXVIII, No. 12, December 1977, p. 47.

#### *Qualified stock option plan for key employees*

Small, innovative companies frequently depend upon stock incentives to attract, and retain, key employees because they cannot afford to pay the high salaries customarily paid by the large firms. Small companies also tend to go through a growth cycle where, in the early stages, technical know-how is the dominant skill required. In due course, commercial products or services are produced from this know-how, but the number of customers remains small. Later, as market opportunities expand and production grows, new requirements develop. The need to manufacture and market products on a larger scale emerges and the need to organize and operate more efficiently begins to rise. This stage requires managerial talents that are oftentimes unavailable in the smaller companies but are plentiful in the larger firms.

The problem for the smaller firm is how it should work to attract more experienced operational managers from the larger companies. Prior to 1976, a widely used and highly successful incentive was the Qualified Stock Option, which allowed a key employee the following choice: if the person chose not to be taxed in the year of grant on the current value of the stock, the person could defer payment of the tax from the exercise date of the option to the earlier of: (1) the year of sale of the underlying stock; or (2) ten years after the grant of the option. The Tax Reform Act of 1976 eliminated this option. As a result, the current law unduly penalizes key employees of smaller firms who must sell their optioned stock at the time of exercising the option in order to pay the required tax. At the same time the individuals are precluded from selling the stock obtained from exercising their options because of the limited or highly illiquid market for such stock.\*\*

That restoration be made of the Qualified Stock Option Plan for Key Employees of small businesses.

#### *Access to capital markets*

Traditionally, small, technically oriented firms have relied on external financing from the public capital markets to support their streams of new products and services that have given vitality and buoyancy to the U.S. economy. In recent years there has been a sharp reduction in the number of firms successfully obtaining funds in the capital markets. The reason is readily apparent to anyone watching the stock market today. Equally illustrative of the venture capital shortage is the recent survey commissioned by the National Venture Capital Association. The report states that in 1975 the bulk of the venture capital industry's investments were in "non" venture businesses. Only 4 percent of the money went to the start-up of new ventures and only 2 percent went to fi-

\*\* "A Program of Tax Revision Proposals to Enhance Capital Formation for Growth Businesses," National Venture Capital Association (NVCA), Washington, D.C. May 1, 1977, pp. 9-11. Also see pp. 34-36 of *Technological Innovation: Its Environment and Management*, U.S. Department of Commerce, Washington, D.C., 1967.

nance first round financings. Both of these figures represented significant declines from previous years.†

To prevent small firms with growth problems from being precluded entirely from the public securities market, the SEC created Regulation A. This regulation facilitates securities offerings of \$500,000 and less, by exempting them from the costly and time-consuming requirements of a full registration. In today's economy, the value of this exemption has been reduced significantly due to inflation. At the same time, the need for increased dollars from the venture capitalists has increased substantially. Both trends emphasize a real need to raise the Regulation A limit to reflect the current realities of our existing capital markets.

Another cause of the current shortage of capital for new ventures is the extreme difficulty of investors to liquidate their venture capital investments once they are made. The basic objective of an investor has always been to realize substantial gains once the venture becomes successful. Not only does this produce a profit that is commensurate with the risk, but it also enables the venture investor to recycle his capital back into other new ventures. If investors cannot realize a profit from their venture capital investments, they will stop making the investments. Then too, gains from successful investments must be sufficient to offset losses, which, in many cases, frequently represent a significant percentage of the total capital invested.

Finally, severe impediments to achieving liquidity have been caused by recent changes in SEC regulations that force investors to urge young and successful innovative firms to seek mergers with larger companies that have broader markets for their shares. This has the counter-productive effect of stifling small promising businesses before they have had a chance to prove they can thrive on their own, let alone making large corporations even larger. In the final analysis, innovation is discouraged and job creation is diminished.

Therefore, liquidity restrictions on venture capital investors should be eased by modifying SEC Rules 144 and 146 so as to facilitate the sale of equities in thriving businesses, as well as the reinvestment of the proceeds in new and growing businesses. Such modifications would be consistent with the needs and protection of both the investor and the securities markets. This would also serve to reduce the liquidation of investments through large corporate takeover.

#### Recommendation 5

That the Security Exchange Commission's Regulation A exemption be increased to include all issues under \$3,000,000 and that SEC regulation procedures for small issues be streamlined; further, that SEC procedures be modified to facilitate the sale of stock in small businesses by major stockholders up to the amount of \$100,000 per year.

† "Statement and Proposals to Promote Liquidity of Venture Capital Investments," NVCA, 1976, p. 1.

#### Pension fund investment

Pension funds provide the primary pool of investment capital today. Their assets are generally estimated to range between \$200 and \$400 billion. The managers of such funds are subject to ERISA regulations. A conservative interpretation of the ERISA regulations requires that the fund managers limit their equity investment to stocks of blue chip firms frequently traded in large volumes on the public exchanges. Therefore, by simply amending ERISA regulations, a new source of funds could be made available to small, technically oriented firms. The Labor Department found considerable merit in the recommendation of a 1976-77 Small Business Administration Task Force on Equity Finance that ERISA be amended in such a way so as to increase the availability of capital to new, small, innovative firms without jeopardizing the safety of pension plan investments.‡ On July 23, 1969, a new regulation went into effect that removed the personal liability of a pension fund manager if a particular investment turned sour, provided the manager had followed department guidelines.\* Although this change will prove beneficial, we believe a further change should be made.

#### Recommendation 6

That ERISA's prudent man standard be restated so that it is clearly applicable to the total portfolio of pension fund investments rather than individual investments; and further, that pension fund managers explicitly be permitted to invest up to 5 percent of pension fund assets in small, technically oriented firms.

#### CONCLUSION

New jobs, especially skilled jobs; better solutions to our national problems of urban decay, pollution, steeply rising costs of food and housing, and health care; and increased competitiveness in international markets—all depend upon our ability to stimulate the rate of technological innovation in the United States. Small businesses are the "well spring" for this innovation and small businesses, in turn, depend upon the availability of capital to sustain them.

The recommendations contained in this paper are suggested as one possible course of action that will lead to increasing the availability of capital through changes in direction and thrust of our corporate tax laws. The changes as recommended would not result in any material loss of revenue to the government and yet would restore the vigor and vitality of our small businesses. Without small businesses we cannot hope to solve some of the economic problems confronting our society today. With them we can ensure our place of leadership in the world economy.

‡ Pages 14 and 15 of the cited report.

\* *Washington Post*, July 15, 1979.

# Corporate venture capital in the computer industry

by KENNETH W. RIND

*Xerox Development Corporation*  
Stamford, Connecticut

and

GENE I. MILLER

*Xerox Development Corporation*  
Los Angeles, California

## INTRODUCTION

Venture capital as practiced by industrial firms differs from conventional venture investing in that motivations beyond strictly financial reward are usually present. Typically, a corporate venture capitalist will be seeking to gain exposure to new markets/technologies, generate new products, develop acquisition candidates, and/or assist a supplier/customer. Corporations also may utilize venture capital concepts in spinning off businesses which are not appropriately kept inside, or in initiating new ventures internally.

While the role of venture capital in the development of the computer industry has been profound from its very inception, in recent years corporations have been playing an increasingly more active role in the financing of new computer-related enterprises. This paper will provide an overall review of the participation of venture capitalists in the computer industry, combined with a description of Xerox' ongoing involvement as a corporate venture capitalist.

## HISTORIC REVIEW OF VENTURE CAPITAL

It is not possible to determine exactly the inception of the organized venture capital industry in the U.S. In fact, groups of domestic and European investors in the late nineteenth and early twentieth centuries were responsible for financing development of several new industries including railroads, steel, petroleum and glass. However, a landmark date for the computer industry was 1911, when a group of wealthy individuals financed and merged three weak companies, International Time Recording Company, Tabulating Machine Company and Computing Scale Company, into a single entity to manufacture and market office equipment. They were wise enough to recruit Thomas Watson as its president in 1914. In 1924 the firm's name was changed to International Business Machines.

Probably the first corporate venturer was Du Pont. When one of its important new customers ran out of funds in 1919, it purchased a 38 percent equity interest and brought in a

new president, Alfred Sloan. General Motors has grown substantially since that investment.

The modern venture capital era is generally considered to have begun after the Second World War, and was given much of its impetus by Laurance Rockefeller, who even prior to that time had helped to finance Eastern Airlines in 1938 and McDonnell Douglas in 1939. Other wealthy family groups, many of whom had originally made their fortunes in earlier ventures, also became active venture capitalists. These included the Phipps (Carnegie Steel) and the Whitneys, J. H. Whitney and his sister Mrs. Joan Payson (heirs of the Vanderbilt fortune).

The formation of American Research and Development (now part of Textron) in 1946 was another landmark event because AR&D was the first venture organization open to public investment and, of course, was the founding investor of Digital Equipment Corporation. Notable also were the 50's financing of eight scientists from Shockley Transistor by Fairchild Camera, and the funding of research into a new copying technique at Battelle by Haloid Corporation, later to change its name to Xerox Corporation.

Also in the 50's, a new spate of companies were founded with government R&D contracting as their major business. Several not-for-profit research groups associated with universities spun-out and prospered in defense-oriented areas. These included: Itek in reconnaissance; GCA in geophysics; Tracor in undersea warfare; and Conductron, now a part of McDonnell Douglas, in radar signal processing.

In the late 1950's and early 1960's a number of successful organizations were formed by large groups leaving the major data processing companies. Examples are: Control Data, founded in 1958 by a group from Univac; Memorex, founded in 1961 by people from Ampex; Scientific Data Systems (later acquired by Xerox), founded in 1964 by Packard-Bell personnel; and Mohawk Data Sciences founded in 1975 by a Univac spin-out. Two successes of this period can be traced to the concept of gathering a number of smaller technological firms under a single corporate wing. Litton Industries, founded in 1952, and Teledyne, founded in 1961, were created in this manner. Other venture capital backed success



stories, such as Digital Equipment and Raychem spun out of non-profit organizations to commercialize new products.

Drawn by these large gains, many new groups entered the venture capital field in the 1960's and 1970's.

- Small Business Investment Companies (SBIC's) were authorized by the Small Business Investment Act of 1958. They are corporations, licensed by the Small Business Administration (SBA), an independent government agency, that are provided with tax incentives and government loans of as much as \$35 million (up to four times the invested capital) to make equity-type investments in small businesses. While 722 SBIC's were licensed, and more than 50 raised public funds, few could be considered unqualified successes, and the number of active SBIC's sank as low as 272 in 1976 before strongly rebounding in recent years.
- A number of new closed-end public venture funds were formed in the late 1960's and early 1970's, among them Inventure Capital, Fund of Letters, Value Line Development Capital, Diebold Venture Capital, Price Capital and Source Capital. Most have since left the business. Insurance companies, banks, mutual funds, university endowment funds, and new private pools, some of them using money from foreign investors (including the Rothschilds), became involved in venture capital. Investment bankers also gathered pools of capital for this purpose. From 1969 to 1972 approximately forty venture capital groups with committed assets of almost \$500 million announced their formation. In the past eighteen months over \$300 million of new monies have been committed to venturing.
- Corporations became active venture capitalists in the 1960's. However, the decline of the market in 1970 brought about the exit of many corporate venture capitalists, including such major names as: Du Pont, Ford, Alcoa, Union Carbide, Northrop, Scott Paper and Singer, as well as some newer venturers such as Memorex, California Computer, Data Products, Boothe, Electronic Memories, Mohawk Data and Applied Magnetics.

Nevertheless, the survivors of start-up during that period include many familiar names, including several minicomputer companies (such as Data General, General Automation, Microdata and Computer Automation), a few peripheral equipment companies (including Storage Technology, Pertec and Centronics), several timesharing companies (such as Tymshare, Comshare and Rapidata) and a considerable number of semiconductor start-ups (such as Intel, Mostek, American Micro-Systems, Intersil and Advanced Micro Devices).

Some corporate-backed computer industry ventures of this era include: Corning in Four-Phase; TRW in Datapoint; Burroughs in Decision Data; Fujitsu and Nixdorf in Amdahl; American Research and Development (Textron) in Documentation; and Computer Machinery in Digital Computer Controls (now respectively part of Pertec and Data General).

Public interest in the market recovered in 1971 and 1972, as shown in Table I, before almost collapsing entirely in 1974 and 1975, driving others from the business. The availability of venture capital funds from Small Business Investment Companies showed a like decline, and surveys of the non-SBIC portion of the venture capital industry, while less complete, indicate a similar pattern of severe cutbacks in 1974 and 1975.

#### PRESENT STATUS OF THE VENTURE CAPITAL INDUSTRY

In recent years investments by venture funds have expanded substantially from an estimated \$300 million in 1976, to \$395 million in 1977 and \$500 million during 1978 (according to Stan Pratt, publisher of *Venture Capital*). Furthermore, there has been a strong influx of additional capital into the hands of new and established venture capital pools, encouraged by recent changes in capital gains rates, Rule 144, ERISA and SBIC rules. It is estimated that the industry now commands about \$3.5 billion, divided:

- Private pools—\$1.3 billion
- SBIC's—\$1.2 billion
- Financial corporations—\$0.5 billion
- Industrial corporations—\$0.5 billion

The figure for industrial corporations includes funds only under direct control. There has also been a growing tendency for corporations to invest in venture pools managed by others.

#### *Corporations increase their direct role*

In the last few years a resurgence of interest in corporate venture capital, fueled by excess corporate liquidity and a relentless toughening of anti-trust oversight, is evident. In addition, the entry of foreign corporations into the field has become a major new factor.

Although financial rewards are usually secondary, cor-

TABLE I.—Small company new issues

Year	Number of Companies	Total Raised (Millions)
1968	358	\$ 745
1969	698	1,367
1970	198	375
1971	248	551
1972	409	896
1973	69	160
1974	9	16
1975	4	16
1976	29	145
1977	30	118
1978	37	206
1979E	60	300

porate venture capital funds that have been run by professionals strictly for maximum return, have generally shown 15-20 percent compounded annual returns. However, corporate venture capital can best be considered as another tool to be used for corporate development and should be coordinated with the acquisition, joint venture and licensing activities of the firm. In addition, some elements of public relations and good corporate citizenship may be present in some corporate venture capital programs, particularly those involved with MESBIC (Minority Enterprise Small Business Investment Company) financing.

On the other hand, many corporations have failed as venturers. A recent survey of corporate venture capital organizations made by Tektronix stated that only 7 percent of corporate venture capital organizations regard themselves as being very successful, with over half not even rating themselves as marginal successes. The success rate could be greatly improved if entrants exercised the same degree of planning as they do in their regular business. The difficulties experienced by a corporation seeking to become a venture capitalist usually arise from one of these sources:

#### **Lack of appropriately skilled people**

A venture capitalist must be entrepreneurially motivated, patient, realistically optimistic, good at negotiation, persuasive and able to evaluate people as well as businesses. Also, he must be more than merely familiar with accounting principles, tax regulations, corporate finance structures, securities analysis, and securities law. Good internal people are generally unwilling to leave a company's mainstream activities even if possessing the appropriate skills. Experienced people from the outside are difficult to attract without special compensation packages.

#### **Contradictory rationales**

A corporate venture capital program may find it difficult to act in the best interests of both the investee company and the parent. For example: if the goal of the venture group is to acquire, then equity financing by others is undesirable; if the rationale is an exclusive marketing arrangement or a preferred supplier role, then the investee's operations may be unduly limited. A desire to have continuous profit increases by the parent is also incompatible with the normal activities of a venture operation.

The entire problem can be exacerbated by an improper reporting structure. For example, having the venture group report to the Vice President of Finance is likely to shift focus to profitability; to the Vice President of R&D to technology; to the Vice President of Corporate Planning to market information, etc.

#### **Legal problems**

A corporate venturer must be extremely careful to organize his activities so that they will not run afoul of conflict

of interest problems, including "fiduciary responsibility" and "corporate opportunity" doctrines. However, several corporations have left the field incorrectly believing that they could not get the strategic benefits they wanted out of a venture activity.

#### **Inadequate time horizon**

A venture activity usually shows its losses and problems early, with the successes taking more time to develop than anticipated. Unless a commitment is made for at least five to ten years, a corporate venture fund generally gets terminated in its early years.

#### *A most active list*

Many corporations have made a single venture capital investment, entered into a "new style joint venture" to obtain access to a unique technology, spun-off a single new entity, or found themselves unwittingly with stock in a customer who was unable to pay his bills. The following list, however, describes those industrial companies which seem to be most active in directly providing venture capital to the computer industry, in addition to *Xerox*.

Exxon is actively investing in order to provide acquisition candidates in the information processing industry (with additional involvements in materials and energy investments).

Textron (American Research & Development) continues to be a major participant in the financing of venture situations.

Continental Telephone has initiated a major new investment program in computers and communications.

General Electric has invested for financial purposes and also has had the most active spin-out program. Spin-outs are made only when it has been decided not to keep an activity going, and the only alternative is liquidation.

Technological spin-outs have also been made by Battelle, Bolt Beranek and Newman and Arthur D. Little.

Fairchild Camera and CTS have invested in customers, and Control Data, Burroughs, NCR and Motorola have invested in suppliers. Teledyne, ATO and Telesciences are recent entrants into venture capital.

Active foreign companies investing for technological reasons include: Northern Telecom, Siemens, Nippon Electric, Cable & Wireless, Konishoroku, BASF, and Fujitsu. Other recent foreign investors in semiconductor companies include: Robert Bosch, Lucas Ind., Jaeger and VDO.

Some of the more recent beneficiaries of this upsurge of corporate investments have included:

- *Computers* - Cray, Tandem, Modular Computer, Apple, Qantel, Magnuson;
- *Data Communications* - Paradyne, Computer Communications, Tran;
- *Telecommunications* - MCI, Valtec, Danray, Digital Telephone;
- *Peripherals* - Data Royal, Silonics, Qume;

- *Terminals* - Applied Digital Data, Digi-log, Ramtek, Threshold Technology;
- *Services* - Quotron, Manufacturing Data Systems, Telenet;
- *Miscellaneous* - Xidex, Computer Products, Quantor.

#### *Corporations should be preferred investors*

Corporate venture capitalists believe they should be preferred investors. In addition to the usual financial and strategic assistance given by conventional venture capitalists, corporations also can offer:

- Assistance in almost all facets of corporate endeavor, e.g., setting up financial systems, qualifying suppliers, meeting government regulations;
- Credibility with customers, banks and other investors both from a technical and financial standpoint;
- Relief, if desired, from the full range of corporate activities, e.g., the corporate investor may take on marketing responsibilities or may license the product;
- Immediate income from an R&D or consulting contract if appropriate;
- Customer interface with an interested party;
- An investor with an infinite lifetime, though his time horizon for profitability will be shorter;
- Additional capital where warranted;
- A merger partner, if and when appropriate;
- A more flexible financing package since return on investment may not be the only criterion.

#### *Selecting a specific partner*

There are several points to consider in selecting a corporate venture capitalist to work with.

##### **Compatibility of goals**

Corporations make venture capital investments for diverse reasons including: assisting potential suppliers or customers, gaining exposure to new technologies/markets, growing possible acquisitions, and obtaining a financial return. The business interests of both parties can either reinforce the possibility of success or may lead to future conflicts.

##### **Longevity**

Many corporate venture groups have been shut down due to lack of success or even shifts in strategy. A failure of continuing support will probably arise at a poor time in the economy for raising funds from others.

##### **People**

If the corporate group is not managed by experienced venturers unnecessary conflicts may develop. Also, there may

be a desire for the staff to return to a career path inside the corporation, thereby requiring continual efforts at education.

##### **Flexibility**

The route necessary for decision-making may be short or tortuous. It is essential for the investee corporation to ensure that crises can be met quickly.

##### **Interference**

Unless the relationship is well-structured, the corporation may attempt to require conventional reporting and staff policies which are inappropriate for a venture situation. Curiosity visits may also be a problem.

##### **Time horizon**

Not all corporate venture capitalists realize the length of time that may be necessary to bring a new business to profitability. If your investors do not react rationally to unforeseen slippage, then the venture will be in substantial difficulty.

##### **Style**

Corporate venture groups, like noncorporate ones, differ in attitudes, approaches and interests. A feeling of sympathy, which should have developed before the investment, is generally extremely helpful to a successful relationship.

## VENTURE INVESTING BY XEROX

The approach taken by the Xerox Corporation is one of providing a supplement to ongoing corporate activities and as a way of understanding new areas of interest. In our "venturing" we seek out and endeavor to work with those entrepreneurs forging ahead in the new, the advancing areas, of technology that will or could impact the business environment in which Xerox is or will be functioning.

To understand the Xerox philosophy of venturing one should read the words quoted by Joseph C. Wilson, the Chief Executive Officer of Xerox when it grew from a \$15 million company to one doing over \$1 billion in revenues, at a John Diebold Lecture at the Harvard Business School in 1969:

"Entrepreneurship, by its nature abhors channels . . . it has tended to be individualistic, innovative, venturesome."

and the entrepreneur is the

". . . one who assumes both the risk and the management of an enterprise, and who hires managers, provides guidelines for their functions, and performs within the organization."

"He is a man who has ideas—basic, germinal ideas, not vagrant thoughts—and who has the daring and the confidence to use them. He is the manager who steps out beyond the confines of a specific area of corporate responsibility. He creates, he pioneers, not just to be different, but simply because *this sort of activity expresses his whole being.*

"Ideas alone are not enough. Those who have them must know what to do with them, and how to translate them into reality."

No one enterprise can claim all the bright thoughts and entrepreneurial talent in existence. By venture investing Xerox expects to keep in contact with this talent elsewhere and thus stay abreast of others' perception of the newest aspects of their technology and thereby gain an understanding of how technology is evolving in selected business areas.

Xerox has had a series of experiences with venture investing. The Corporation has participated in this industry either on a part-time basis or with a combination of internal personnel and outside consultants who jointly located, negotiated, and followed the investments made. Those historic investments were generally related, directly or indirectly, to some of the business areas in which the Corporation was engaged. In one case in particular the investee company proved quite successful. In other situations the association was not as rewarding. Near the end of 1975 the conclusion was reached to reformulate the venture investment activity and place it in closer proximity to the long range corporate development function. Collectively, these operations became the Xerox Development Corporation ("XDC"), a wholly-owned subsidiary of the Xerox Corporation.

XDC is charged with the responsibility for identifying opportunities and growth areas for Xerox beyond its present business thrust. Since its founding it has been engaged in acquisitions, venture investments, divestitures, licensing of technologies, and helping to shape this strategic direction the company intends to take in the future. XDC was instrumental in establishing XTEN, a plan now before the Federal Communications Commission which, if approved, would set aside a band of radio frequencies for document distribution, data transmission and teleconferencing.

The early months of XDC were spent gathering its full-time professional staff. Those professionals engaged in venture investments have come from the venture capital community. They are experienced in understanding and relating to entrepreneurs and the early cycles of a new enterprise. The close working proximity to the other members of the XDC group, some of whom are drawn from internal sources, provides a ready means of identifying and gaining access to corporate resources and quicker acceptance of the function and its goals.

XDC is organized in the style of a partnership. Rather than call the members "Partner" they are referred to as "Principal" to reflect that the organization is the wholly-owned subsidiary of its parent the Xerox Corporation.

The individuals engaged in venture investing are themselves organized as a "little" partnership. New opportunities are regularly discussed as are the analyses of investment candidates, and the progress of the enterprises already invested in. This group review is the first point in the invest-

ment process. This little partnership is also the staff responsible for aiding the health of the investee companies and guiding the relationships that may develop with the Corporation.

Should an investment opportunity appear promising, other elements of the "larger" partnership are brought in to assist in the further study of the enterprise. Collectively the various aspects and interrelationships of a potential investment are threshed out and understood from the alternative viewpoints. With that expanded understanding a decision is reached by the venture investment group to proceed or not, and if so on what terms and conditions. Should the decision to proceed be made and satisfactory terms negotiated then final approval, on all except the largest of investments, is made within XDC by the concurrence of the venture investment group and the Chairman of XDC. Large investments are approved, in addition, by the Chairman and Chief Executive Officer of Xerox.

The principal criteria against which a new investment opportunity is measured at Xerox are: (1) the entrepreneurial quality of the management, (2) the nature and relevance of the enterprise to the current or prospective business environment of the parent Corporation, and (3) the potential for building a significant business.

Most industrial venture groups espouse similar measures. What often tends to happen is that the second criterion mentioned—relevance of technology to the parent entity—drives out or reduces consideration of the other points. It is here that the professional staffing and the mind set and instincts of that staff become increasingly significant.

If the long-standing orientation of the people making the investment decision is to relate to entrepreneurs and to build a profitable business the analytical approach and the attractiveness of situations will not easily or naturally be directed toward technical interest as a basis of advancing funds.

Certainly that orientation must be modified (hence the value of relating to others in corporate long-range planning and mergers and acquisitions), but it should not be so changed as to lose sight of the fundamental nature of business—to serve the needs of the marketplace. The ultimate reasons for venturing by an industrial entity—to open alternative channels in order to accomplish a given task; to relate to those aspects of its environment which the parent lacks the resources (men, material, motivation, time, or money) to accomplish itself or within its own structure; and to be open to new ideas from others—mandate commercial success. It is only appropriate therefore that the probability of commercial success be a significant criterion. Such probability is derived from an analysis of the proposed business plan.

The definition of what falls within the area of relevance to the parent Corporation is a subjective judgment. For our own part we have attempted to understand the businesses the Corporation is in, best summarized as information handling systems, and project the changes that will or could occur. Further, we have endeavored to identify the subsystems that are the building blocks of those larger systems and the impact they have on the total configuration. The sum of this body of knowledge lets one see how various suggestions

relate, or could relate, to the overall businesses of Xerox. Specifically what this means is that oil wells and real estate are out of consideration; and memory devices, electronic components, and communications equipment are in for consideration.

What is often confused with criteria—the specific information that should be provided or will be requested in evaluating a given situation—should not be overlooked. There is no absolutely correct way to prepare a business plan. Several excellent discussions on the preparation of a business plan are available in most libraries. What the plan and the general information made available will do is provide a means of measuring management and evaluating their skills and strategies. The plan will show: (1) the depth of understanding of the particular business; (2) a sense of organization and direction; (3) a perception of the marketplace, its needs, and means and timetable for meeting those needs; (4) the risks and problems in reaching the market with a product successfully.

Accordingly the following should be provided, described or otherwise explained:

- a market definition and survey—what are the users' needs, what are their reactions to the product, who are the competitors and what are their capabilities, and estimated share potential for the enterprise;
- the product—what are its qualities, its life expectancy, how does it meet user needs, and how does it compare to competitive offerings;
- the operating plan—including timetable, achievement milestones, and manning requirements;
- the financial plan—how are the financial needs of the enterprise to be met;
- management—background, reputation, past performance.

Personal contact in conjunction with the plan will answer some of the following issues regarding management: (1) the level of drive/motivation to achieve; (2) resourcefulness in meeting and dealing with the unexpected; (3) credibility as a leader; (4) judgment—the ability to identify significant

milestones in measuring the company's progress and act according to achievement against those milestones.

Being the venture investing affiliate of an industrial organization brings its own special sensitivities which must be kept in mind in relating with other businesses as a minority owner. These areas of concern are associated with the pragmatic means of building a rapport and the mechanics of the formal investment. In structuring a relationship there is no such thing as a typical investment for us at Xerox. Each opportunity is studied on its own merits. Each negotiation is tailored to the facts of the particular situation. Each investment is followed with a recognition of, or attempt to recognize, the individual characteristics of the industry, the people involved, and the inherent "ups and downs" of any new enterprise.

For our own part Xerox does not seek a venture investment situation wherein it will play the dominant role or attempt to control the course and direction of the enterprise. This is not to say we will not give advice or assistance. It does say we believe strongly in the entrepreneur. That the entrepreneurial spirit is essential to innovation and the resourcefulness to deal quickly and successfully with the unexpected. To take control and eliminate the incentives for success would greatly suppress that spirit and defeat the initial purpose of the investment.

We do not take seats on the Board of Directors of investee companies. We do negotiate the right to visit at reasonable times and to attend Board meetings. A provision of the agreement will provide for regular financial reporting.

The investment is not a prelude to acquisition by Xerox. This may ultimately come about; to date it hasn't. If such a turn of events materializes it will be because the entrepreneurial management wants such a merger, not because it has been mandated with the original funding.

The Xerox approach to venture investing is the development of a mutually rewarding relationship. This has meant for us carrying on our activities in the professional manner of the traditional venture capitalist while building channels of understanding between the Corporation and the investee company. It is working well for us. It is predicated on finding the entrepreneur working in an interesting area and supporting his activities. From that, if more is to develop, the parties are mutually free to chart their directions.

# Structured procedure for comparison and selection of computer system designs

by ANTONIO VALLONE

Computer Sciences Corporation  
Silver Spring, Maryland

Decisions about selecting a configuration for a computer system require an unbiased comparison among alternative designs of the system. Several heterogeneous factors need to be considered and their combined effect must be evaluated. The procedure provides a structure to the selection process activities: developing a selection plan, evaluating each design, and ranking the alternatives. It is based on a cost-effectiveness methodology which characterizes each design by the life-cycle cost through a "system cost index" and by the design effectiveness in reaching the system objectives through a "system utility index." The procedure is applicable to the selection of a system, to tradeoff analysis during the system design, and may constitute the framework for the analysis of the risk associated to a design implementation.

## INTRODUCTION

The acquisition of an information system constitutes a painstaking problem for a manager of an organization that makes substantial investments in computer systems [1], [2]. He should identify the needs of the actual and potential users of the system, define the scope of the system, and specify the essential and optional requirements and capabilities for the system. When he has done all his homework his problems are not ended. In fact, he will have to decide which system to implement among several proposals of system designs. He is flooded with a quantity of data representing heterogeneous parameters of system performance and cost from which he has to derive a simple "number" representing the "best" design.

Practically the only method presently existing to help the decision maker is to translate all the data in monetary terms [3]. This method is largely employed and constitutes a teaching topic in many administration courses. The difficulty encountered in several cases is in the definition of a reasonable monetary value for the system parameters when the system objectives are not immediately or conveniently related to economic benefits. Such are the cases of computer systems directly devoted to technical or scientific analysis or dedicated to information processing for decision analysis. The economical benefits are remote and often the very existence

of the computer system is needed to assess its benefits (e.g. information value).

This paper presents a structured procedure for selecting a system configuration among alternative designs based on a cost-effectiveness methodology. The methodology is an extension of E. O. Joslin's concepts [3] through the application of the utility curves concepts introduced by D. Hurta [4] in the risk analysis area.

The procedure has been developed for NASA-GSFC [5], [6], and is applicable to the selection of an entire information system as well as a subsystem, to trade-off analysis during system design, and it is also the framework for the analysis of the risk associated with the decision to implement the selected design.

## COST-EFFECTIVENESS METHODOLOGY

The methodology subdivides the items related to the selection of the computer system in two classes: the cost elements which directly affect the life-cycle cost of the system, and the selection factors which are traceable to the system objectives. Accordingly, each system design is represented by two indices: the system cost index that aggregates the time distribution of all cost elements, and the system utility index that expresses the effectiveness of the design in fulfilling the system objectives. The two indices combine through a ranking algorithm in a system rank index which represents the cost-effectiveness of the design.

The definition of the system cost index is directly obtained by discounting to a reference "present time" the costs incurred during the system life cycle (Table I) by means of a specific value (or a range of values) of the interest rate.

The basic model to compute the system cost index from expenses distribution and interest rate value is the "present value of expenditures" (PVE) represented by the equation

$$PVE = \sum_{K=0}^n \frac{CE_K}{(1+r)^K}$$

which is easily implemented in a computational algorithm [6]. In the equation,  $n$  is the length of the system life-cycle usually in years,  $r$  is the interest rate value, and  $CE_K$  the

TABLE I.—Life Cycle Cost Elements

LIFE-CYCLE COST ELEMENTS (EXAMPLE)	
0. SYSTEM DESIGN	6. OPERATION
1. HARDWARE	6.1 MAINTENANCE (H/W & S/W)
1.1 MAINFRAME SYSTEM	6.1.1 PREVENTIVE MAINTENANCE
1.2 PERIPHERAL SYSTEM	6.1.2 CORRECTIVE MAINTENANCE
1.3 SPECIAL HARDWARE	6.2 SYSTEM EXPANSION & ENHANCEMENT
2. SOFTWARE	6.2.1 HARDWARE
2.1 SYSTEM SOFTWARE	6.2.2 SYSTEM & APPLICATION S/W
2.2 APPLICATION SOFTWARE	6.3 PERSONNEL
3. COMMUNICATION	6.3.1 SYSTEM OPERATIONS
3.1 NETWORK EQUIPMENT	6.3.2 APPLICATION OPERATIONS
3.2 NETWORK LINES	6.3.3 SUPPORT
4. FACILITIES	6.4 UTILITIES
4.1 SPACE	6.5 POSSIBLE REVENUES*
4.2 POWER SUPPLY	7. END OF LIFE CYCLE
4.3 AIR CONDITIONING	7.1 SYSTEM DISPOSAL COST
5. TRANSITION	7.2 SYSTEM RESIDUAL VALUE*
5.1 PERSONNEL TRAINING	
5.2 SOFTWARE	

\*REPRESENTED BY A NEGATIVE COST

aggregation of expenditures (and revenues) occurring during the Kth period of the life-cycle.

The system utility index aggregates the heterogeneous selection factors in a measure of system effectiveness through the definition of a utility curve and a weight [5] for each selection parameter (Table II) associated to the selection factors. The utility curve (Figure 1) measures how better (or worse) is any level reachable by the selection parameter with respect to a specific nominal level. The weight measures the

TABLE II.—Selection Factors and Parameters (example)

SELECTION FACTORS	SELECTION PARAMETERS
ACCURACY	<ul style="list-style-type: none"> <li>● COMPUTATIONAL ERROR (ABSOLUTE OR RELATIVE)</li> <li>● NUMBER OF TERMS IN A SERIES EXPANSION</li> <li>● NUMBER OF BITS FOR NUMERICAL REPRESENTATION</li> </ul>
RESPONSE TIME	<ul style="list-style-type: none"> <li>● TRANSACTION RESPONSE TIME (FOR EACH TRANSACTION)</li> <li>● EVENT RESPONSE TIME (FOR EACH EVENT)</li> <li>● AVERAGE RESPONSE TIME (FOR EACH CLASS OF TRANSACTIONS OR EVENTS)</li> </ul>
THROUGHPUT	<ul style="list-style-type: none"> <li>● NUMBER OF PRODUCTION JOBS PERFORMED DURING A WORKING PERIOD (FOR EACH JOB CLASS)</li> <li>● TURNAROUND TIME FOR JOBS SUBMITTED DURING A WORKING PERIOD (FOR EACH JOB CLASS)</li> <li>● VOLUME OF DATA PROCESSED (FOR EACH CLASS)</li> </ul>
UTILIZATION	<ul style="list-style-type: none"> <li>● PERCENT OF TIME A RESOURCE IS ALLOCATED DURING A WORKING PERIOD (FOR RESOURCE CLASS)</li> <li>● PROBABILITY OF RESOURCE SATURATION DURING A WORKING PERIOD (FOR RESOURCE CLASS)</li> </ul>
EASE OF TRANSITION	<ul style="list-style-type: none"> <li>● IMPLEMENTATION TIME FOR A FUNCTION (FOR EACH CLASS OF FUNCTION)</li> <li>● CURRENT SYSTEM DOWNTIME</li> </ul>
AVAILABILITY	<ul style="list-style-type: none"> <li>● PERCENT OF TIME A FUNCTION IS OPERATIONAL DURING A WORKING PERIOD (FOR EACH CLASS OF FUNCTIONS)</li> <li>● <math>MTBF/(MTBF + MTTR)^{11}</math></li> </ul>
RELIABILITY	<ul style="list-style-type: none"> <li>● PROBABILITY THAT A FUNCTION PERFORMS SUCCESSFULLY (FOR EACH CLASS OF FUNCTION)</li> <li>● MTBF</li> </ul>
FLEXIBILITY	<ul style="list-style-type: none"> <li>● TIME REQUIRED TO RESPOND TO SUCH ANOMALOUS CONDITIONS AS WORKLOAD INCREASE OR FAILURE</li> </ul>
EASE OF USE AND/OR DEGREE OF AUTOMATION	<ul style="list-style-type: none"> <li>● TIME REQUIRED BY THE USER TO ACCESS A SYSTEM FUNCTION (FOR EACH USER AND FUNCTION CLASSES)</li> <li>● TIME REQUIRED BY THE SYSTEM OPERATOR TO SET UP A FUNCTION (FOR EACH FUNCTION CLASS)</li> </ul>
EXPANDABILITY	<ul style="list-style-type: none"> <li>● TIME REQUIRED TO INCREASE THE SYSTEM'S CAPABILITIES</li> </ul>
MAINTAINABILITY	<ul style="list-style-type: none"> <li>● TIME REQUIRED TO FULLY RESTORE A FUNCTION (FOR FUNCTION CLASS)</li> <li>● TIME REQUIRED TO DETECT AND REPAIR A FAILURE</li> <li>● TIME REQUIRED TO CORRECT FAILURE</li> </ul>

<sup>11</sup>MTBF = MEAN TIME BETWEEN FAILURES  
MTTR = MEAN TIME TO REPAIR (OR RECOVER) THE FAILURE

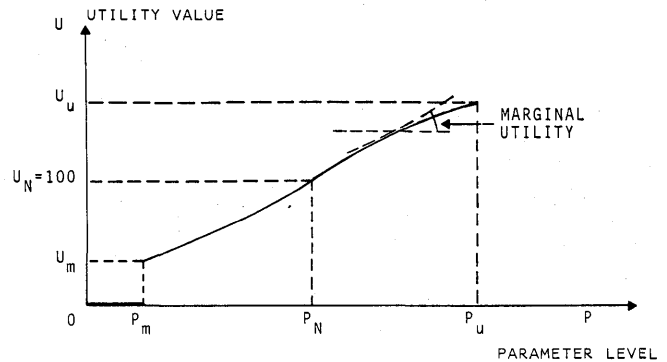


Figure 1—Utility curve.

relative importance of each parameter with respect to the system objectives.

Several interpretations may be associated to the utility curve concept. If the system objectives have a purely economical nature, the utility curve may represent Joslin's relative monetary worth of each level of a parameter [3]. A probability related meaning is possible when the system life stretches quite into the future. The utility curve translates the probability that each level of a parameter will fulfill the uncertain future objectives. In general, the utility curve is the way to accommodate subjective preferences and judgments within the selection process through a quantitative representation.

The aggregation of the various parameter utilities generates the system utility index. An example of aggregation model is [6]:

$$U_s = U_E(W_E + W_o U_o)$$

In the model,  $U_E$  is the geometric average of the essential parameter utilities,  $U_o$  is the arithmetic average of the optional parameter utilities, and  $W_E$  and  $W_o$  the respective aggregated weight.

The system cost index and the system utility index combine in a system rank index by means of a general dominance relation [5]. Also, a ranking algorithm can be defined for a specific class of computer systems. For example, if Grosch's Law [7] can be assumed for large systems, the system rank index is defined by

$$R_s = (U_s)^{1/2}/C_s$$

and the system design with the largest value of  $R_s$  will be selected.

## THE SELECTION PROCEDURE

The process of selecting a computer system encompasses several activities such as defining the system's important features, evaluating performance parameters, and ranking system designs with respect to some selection criteria.

The various activities needed to apply the cost-effectiveness methodology are integrated in a procedure that employs, as a starting point, the objectives and requirements

specified for the system (and also employed by the design activity) and that systematically evaluates each alternative design to identify the best design for the system.

Broadly speaking, the procedure is composed of three major steps: developing a selection plan; evaluating the selection criteria for each alternative configuration design; and determining the order of preference among designs, based upon how closely they satisfy the selection criteria.

Figure 2 presents a scheme of the selection procedure.

Defining the selection plan is the critical step of the procedure because it must provide subsequent activities with the inputs and the controls that will ensure consistency and objectivity within the selection process. Consistency is attained by specifying the methods that will be employed to evaluate each design. Objectivity can be attained by defining in advance topics and criteria related to subjective assessments so that each design will be treated in the same way.

The selection plan must contain an accurate description of the selection process taking into account that the effort dedicated to the selection process should be commensurate with the expected potential saving. The principal functions of the selection plan are: identify the system selection factors and the system life-cycle cost elements; generate the selection parameter utility; and specify models and tools to be employed for evaluating effectiveness and cost and for assessing the alternative design ranking.

Cost elements and selection factors have been already illustrated in Tables I and II respectively.

To each selection parameter identified, correspond a utility curve and a weight generated through: a definition of the

acceptable range of the parameter; an assessment of the parameter weight and the utility values for two or more parameter levels; and a calibration of the utility curve model. The first two activities are performed by one or more respondents who have a good knowledge of the system objectives, of the system requirements, and of how a variation in the level of each selection parameter may affect the fulfillment of the system objectives. The last activity will ensure consistency of assessment both among the respondents and among the various utility curves. The choice of the models depends on the specific interpretation of the utility curves. Details may be found in [5] and [6].

For each selection parameter and cost element, the selection plan should define a model and/or an algorithm to evaluate the parameter or cost level and should specify how the model will be employed for any expected system design. For instance, system performance parameters may be estimated by models of the flow of data, controls, and activities within the system such as analytical algorithms based on queuing theory or simulation programs. Examples of simulation programs are the Multi-Purpose System Simulator (MPSS) [8] and the Data System Dynamic Simulation (DSDS) [9]. The selection plan should specify types, volumes, and frequencies of data and information (i.e., the test workload) that enter the system. Other selection parameters, such as flexibility and ease of use, may be estimated through subjective judgments if adequate algorithms are lacking. The selection plan should specify the guidelines, procedures, and methodologies to be used in the subjective assessments in order to ensure a high level of objectivity and consistency.

For the system cost index, the selection plan should specify the following items: time span and present time of the system life cycle; methods and procedures to compute and distribute each cost element; and value of the yearly interest rate or cost of money.

The selection plan should also specify how to handle exceptional cases, such as proposed design options and/or alternative contractual conditions. In general, each alternative should be considered as generating a different system design in order to assess the global and marginal effect of each alternative.

The development of the selection plan should be done with enough care so that the subsequent evaluation and ranking steps may be routinely performed on each alternative design. These last two steps will be performed when each design has been completely defined and will comprehend, in general, the following activities:

- Analyze the documentation of each system design.
- Model the designed system configuration to represent both the selection parameters and the cost elements.
- Estimate the level of each selection parameter, as well as the amount and time distribution of the expenditures relative to each cost element.
- Compute
  - the system design utility index using the specified utility curves and algorithms

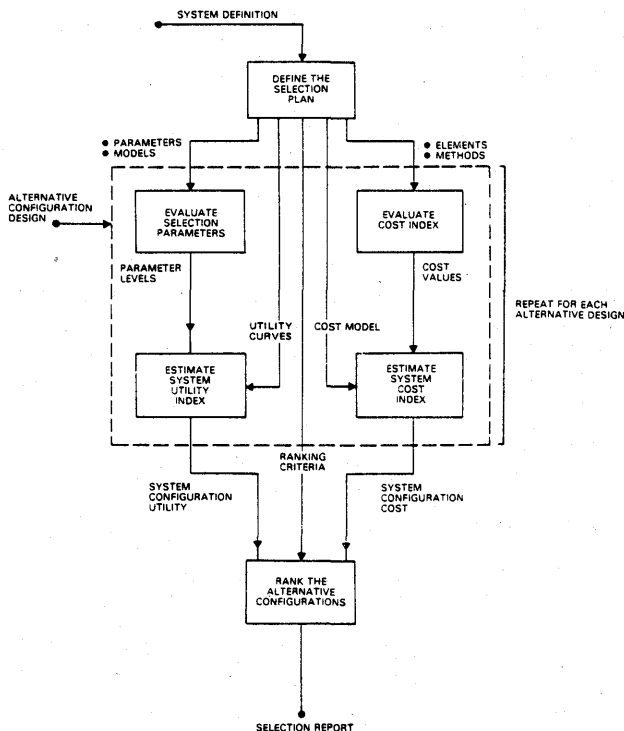


Figure 2—Scheme of the selection procedure.



- the system design cost index using the specified models
- the system ranking index using the specified model.
- Order all alternate designs according to the ranking index.

Analysis of the design documentation is the critical activity because it must extract the data and information required to evaluate the design according to the selection plan specifications, and identify all exceptional conditions. The remaining activities should follow the selection plan strictly and will not be discussed further here since they have been analyzed in detail in [5] and [6]. For illustration, the appendix presents a hypothetical example.

## CONCLUSION

The selection of a computer system design among several alternative configurations requires the analysis of heterogeneous items, both technical and economic, that need to be combined in a consistent and rational ordering of the alternatives.

The selection procedure presented in this paper is based upon a cost-effectiveness methodology that subdivides those items in two classes: selection parameters representing the ability to fulfill the system objectives; and cost elements affecting the life cycle cost of the system. The impact of each class on the selection is combined in a selection criterion: the system utility index as a measure of the system design effectiveness; and the system cost index as a measure of the distribution of costs over the system life. The two system indices are evaluated for each alternative system design which is thus characterized by a pair of utility costs. The application of a ranking algorithm provides a rational ordering of the designs from which the "best" system can be selected.

The various activities of the selection process are structured in a systematic procedure that clearly identifies the functions, the inputs and the results of each step. Furthermore, the procedure can ensure a maximum degree of consistency and objectivity through the development of a selection plan which specifies in advance any controlling topic and subjective assessment needed for the evaluation of the alternative designs. The selection procedure has been experimentally applied with success to a real case study [6].

## REFERENCES

1. Timmreck, E. M., "Computer Selection Methodology," *ACM Computing Surveys*, 5 (Dec. 1973).
2. King, J. L. and Schrems, E. L., "Cost-Benefit Analysis in Information Systems Development and Operations," *ACM Computing Surveys*, 10 (Mar. 78).
3. Joslin, E. O., *Computer Selection*, The Technology Press, Inc., Fairfax, Va., 1977.
4. Dr. Hurta, "Risk Assessment Methods," (personal communications).
5. Vallone, A. and Bajaj, K., *Cost-Effectiveness Methodology for Computer System Selection Part I* CSC/TR-79/6850, Contract NAS 5-20640 (TA) 571-060, NASA-GSFC, Greenbelt, Md.
6. Vallone, A. and Bajaj, K., *Cost-Effectiveness Methodology for Computer System Selection Part II*, CSC/TR 79/6852, Contract NAS 5-20640 (TA) 571-060, NASA-GSFC, Greenbelt, Md.
7. Cale, E. G., Gremillion L. L., and McKenney, J. L., "Price/Performance Patterns of U.S. Computer Systems," *Comm. of ACM*, Vol. 22, No. 4, (Apr. 1979).
8. CSC/SD-79/6852 MPSS User's Guide, Contract NAS 5-20640 (TA) 571-060, NASA-GSFC, Greenbelt, Md.
9. *Data System Dynamic Simulation (DSDS)-User Manual*, General Electric Co., Contract NAS 8-31532, NASA-MSFC, Huntsville, Ala. 1978.

## APPENDIX—HYPOTHETICAL EXAMPLE OF THE SELECTION PROCEDURE

In order to furnish a complete description of the procedure, this Appendix describes a hypothetical example of system selection taken from a presentation of the procedure to NASA-GSFC. The case study is oversimplified, to cover completely the selection procedure in a short space.

The example is the selection on an hypothetical entry system to support program coding and debugging for a software development facility.

A series of tables and figures describe the example with the following types of information:

- Definition of the system (Table A-I)
- Selection plan (Table A-I) and utility curves (Figures A-1 through A-3)
- Description of the alternative configuration designs (Table A-III)
- Alternative design evaluation and ranking:
  - selection parameter evaluation (Table A-IV)
  - system utility index determination (Table A-V)
  - life-cycle cost and system cost index evaluation (Table A-VI)
  - utility index-cost index diagram (Figure A-4)
  - system rank index evaluation and identification of the best design (Table A-VII)

Although the tables and figures are self-explanatory, some discussion of the content will aid in following the example. The system is defined by specifying its principal characteristics (Table A-I): system objectives, functions, and nominal requirements specifications. Obviously these are only a

TABLE A-I.—Definition of System

SYSTEM DEFINITION	
● OBJECTIVES:	PROVIDE COMPUTER SUPPORT TO PROGRAM CODING
● FUNCTIONS:	GENERATE SOURCE CODE COMPILE GENERATED CODE CORRECT SYNTAX ERRORS
● SPECIFICATIONS:	NUMBER OF PROGRAMMER POSITIONS SUPPORTED 10 NOMINAL (CRT TERMINALS)
	COMPILE TURNAROUND TIME 15 MIN NOMINAL (AVERAGE OVER A MIXTURE OF PROGRAMS)
	EDIT RESPONSE TIME 10 SEC NOMINAL (AVERAGE OVER A MIXTURE OF EDIT COMMANDS)

TABLE A-II.—Selection Plan

DEVELOPMENT OF THE SELECTION PLAN	
•	SELECTION FACTORS: IDENTICAL TO THE SYSTEM SPECIFICATION PARAMETERS
•	COST ELEMENTS: <ol style="list-style-type: none"> <li>1. PROGRAMMER POSITIONS                             <ol style="list-style-type: none"> <li>1.1 HARDWARE</li> <li>1.2 SUPPORT SOFTWARE</li> </ol> </li> <li>2. TERMINALS - MAINFRAME COMMUNICATION NETWORK</li> <li>3. H- W &amp; S/ W MAINTENANCE (10% OF COST/YEAR)</li> <li>4. SYSTEM ANALYST SUPPORT TO PROGRAMMERS (5% OF PROGRAMMERS @ \$30,000/ YEAR PROGS)</li> </ol>
•	GENERATION OF UTILITY CURVES FOR EACH SELECTION PARAMETER
•	SYSTEM UTILITY INDEX: GEOMETRIC AVERAGE OF EVALUATED PARAMETER UTILITIES
•	SYSTEM COST INDEX: PRESENT VALUE OF EXPENDITURES $I_0 = 10\%$ $n = 5 \text{ YEARS}$
•	SYSTEM DESIGN EVALUATION: <ul style="list-style-type: none"> <li>- MODEL SYSTEM DESIGN WITH MPSS</li> <li>- NORMAL WORKLOAD OF MAINFRAME</li> <li>- PROGRAM SOURCE DISTRIBUTION</li> <li>- EDIT DISTRIBUTION</li> </ul>
•	RANKING MODEL <ul style="list-style-type: none"> <li>- UTILITY PER DOLLAR</li> <li>- ALGORITHM:                             <math display="block">RS = \frac{C}{U \cdot S}</math> </li> <li>- SELECT THE MINIMUM RS</li> </ul>

small subset of what should have been needed to define an actual system. The nominal requirement levels are assumed to derive from a compromise between user need to optimize programmers' performance and constraints due to the hypothetical operational environment.

The selection procedure follows the steps discussed in the previous sections.

First, the selection plan is developed (Table A-II) using the system definition. The major topic is represented by the generation of the selection parameters' utility curves and weights (see Figures A-1 through A-3). These have been assessed by the writer through an arbitrary judgment. The rationale behind the assessment of parameter ranges and utility values derives from a hypothetical analysis of the programmers' productivity.

The system's hypothetical design, which does not belong to the selection procedure, resulted in several alternative system designs. These are grouped (Table A-III) in two basic configurations. The first one relies upon the host central

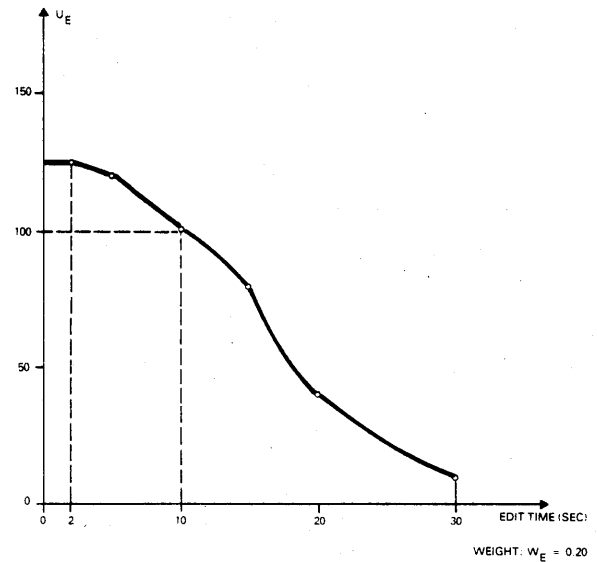


Figure A-2—Utility of edit time.

computer for the editing capability; the second configuration distributes the editing capability. Each of the two configurations is associated with three alternative communication solutions. Therefore, six designs need to be evaluated.

The evaluation activities for the selection procedure are described next. First, each selection parameter is evaluated for each of the six designs (Table A-IV). Response time and turnaround time, hypothetically derived from the modeling and simulation of each design, have been assessed in actuality by a subjective judgment of the writer. Combining parameter levels and utility curves by means of the system utility model provides the value of the system utility index for each system design (Table A-V). This value represents the capability of each design to meet the system objectives. Note that only design D is above the nominal level. However, the other designs are also technically acceptable and constitute a trade-off among requirement specifications.

The second evaluation concerns the system life-cycle cost of each design (Table A-VI). Each cost element specified by the selection plan is evaluated from a hypothetical vendor price list or, for maintenance and personnel support, from the corresponding models also specified by the selection

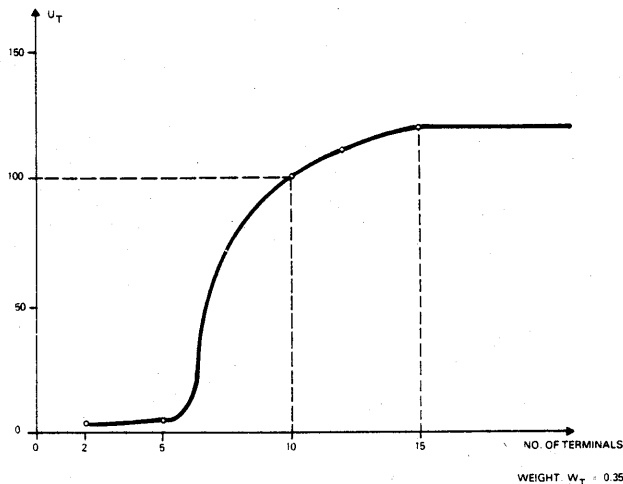


Figure A-1—Utility of number of terminals.

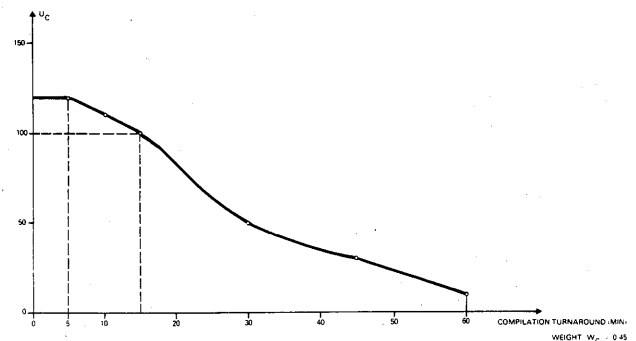


Figure A-3—Utility of compiler turnaround.

TABLE A-III.—Description of Alternative Configuration Designs

ALTERNATIVE CONFIGURATION DESIGNS		SYSTEM DESIGN
1. PASSIVE CRT TERMINALS		
1.1 DIRECT CONNECTION	10 TERMINALS	A
1.2 USE OF REMOTE CONCENTRATORS		
1.2.1 1 CONCENTRATOR	6 TERMINALS	B
1.2.2 2 CONCENTRATORS	12 TERMINALS	C
2. DISTRIBUTE EDITING CAPABILITY		
2.1 INTELLIGENT CRT TERMINALS DIRECTLY CONNECTED 10 TERMINALS		
2.2 USE OF FRONT-END MINIS		
2.2.1 1 MINI	8 TERMINALS	E
2.2.2 2 MINIS	15 TERMINALS	F

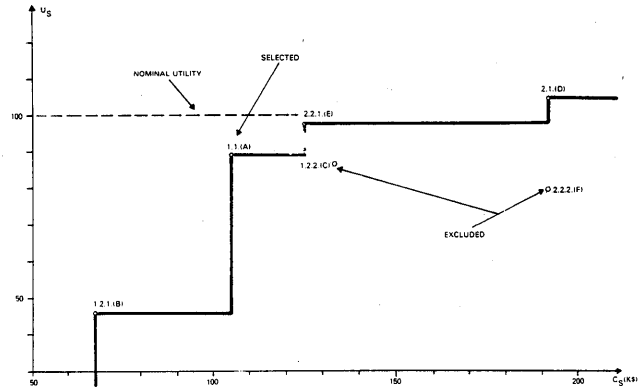


Figure A-4—Utility index—cost index diagram.

TABLE A-IV.—Selection Parameter Evaluation

CONFIGURATION SELECTION PARAMETERS	1.1 (A)	1.2.1 (B)	1.2.2 (C)	2.1 (D)	2.2.1 (E)	2.2.2 (F)	NOMINAL
NUMBER OF TERMINALS	10	6	12	10	8	15	10
COMPILE TURNAROUND (MINUTES)	20	10	25	15	12	30	15
EDIT RESPONSE TIME (SECONDS)	15	12	12	2	5	5	10

TABLE A-VII.—System Rank Index Evaluation

SYSTEM DESIGN	U <sub>S</sub>	C <sub>S</sub>	R <sub>S</sub> <sup>1</sup>	RANK
1.2.1. (B)	46	65.8	1.43	3
1.1. (A)	89	105.1	1.18	1*
2.2.1. (E)	99	125.5	1.27	2
2.1. (D)	105	192.0	1.83	4

<sup>1</sup> SYSTEM RANK INDEX COMPUTED WITH THE MODEL:

$$R_S = \frac{C_S}{U_S}$$

\* SELECTED SYSTEM CHARACTERISTICS:

- DIRECT CONNECTION
- 10 TERMINALS (NOMINAL 10)
- 20 MIN COMPILE (NOMINAL 15 MIN)
- 15 SEC EDIT (NOMINAL 10 SEC)

TABLE A-V.—System Utility Index Determination

PARAMETER WEIGHT	CONFIGURATION SELECTION PARAMETERS	CONFIGURATION						NOMINAL
		1.1 (A)	1.2.1 (B)	1.2.2 (C)	2.1 (D)	2.2.1 (E)	2.2.2 (F)	
0.35	NUMBER OF TERMINALS	100	10	110	100	80	120	100
0.45	COMPILE TURNAROUND	85	110	68	100	105	50	100
0.20	EDIT RESPONSE TIME	80	95	95	125	120	120	100
1.0	SYSTEM UTILITY INDEX <sup>1</sup>	89	46	86	105	99	81	100

<sup>1</sup> THE SYSTEM UTILITY INDEX IS COMPUTED AS A WEIGHTED GEOMETRIC AVERAGE OF THE SELECTION PARAMETERS UTILITY VALUES:

$$U_S = (U_T)^{WT} (U_C)^{WC} (U_E)^{WE}$$

$$WT + WC + WE = 1$$

TABLE A-VI.—Life-Cycle Cost and System Cost Index Evaluation

COST ELEMENTS	CONFIGURATION						
	1.1 (A)	1.2.1 (B)	1.2.2 (C)	2.1 (D)	2.2.1 (E)	2.2.2 (F)	
POSITION: H/W <sup>1</sup>	30.0	18.0	36.0	90.0	44.0	60.0	
S.W. <sup>2</sup>				3.0	10.0	10.0	
COMMUNICATIONS LINE EQUIPMENT (K\$)	5.0	2.0	4.0	5.0	4.0	8.0	
TOTAL ACQUISITION (K\$)	35.0	23.0	46.0	98.0	58.0	78.0	
MAINTENANCE (K\$/yr)	3.5	2.3	4.6	9.8	5.8	7.8	
PERSONNEL SUPPORT (K\$/yr)	15.0	9.0	18.0	15.0	12.0	22.5	
TOTAL OPERATION (K\$/yr)	18.5	11.3	22.6	24.8	17.8	30.3	
PVE OPER. <sup>3</sup> (K\$)	70.1	42.8	85.7	94.0	67.5	114.9	
C <sub>S</sub> <sup>4</sup> (K\$)	105.1	65.8	131.7	192.0	125.5	192.9	

<sup>1</sup> PASSIVE TERMINALS @ 3.0 K\$  
INTELLIGENT TERMINALS @ 3.0 K\$  
MINICOMPUTER SYSTEM @ 30.0 K\$

<sup>2</sup> EDIT S/W FOR TERM. 3.0 K\$  
SYSTEM S/W FOR MINI 10.0 K\$

<sup>3</sup> PVE OPER. = a(r,n) \* TOTAL OPER.  $a(r,n) = \sum_{k=1}^n (1+r)^{-k}$   
r = 10%; n = 5(yr); a(r,n) = 3.7907867(yr)

<sup>4</sup> C<sub>S</sub> = PVE OPERATION + TOTAL ACQUISITION

plan. The computation of the system cost index is a direct application of the formula relative to the PVE.

The last evaluation concerns the system ranking index for each design. The first step consists of plotting the system utility index (U<sub>S</sub>) versus the system cost index (PVE) for each design (Figure A-4). This step identifies the designs (namely C and F) to exclude immediately because of inferior cost-effectiveness. With the second step, the remaining designs are then ordered according to the system ranking index computed by means of the model specified by the selection plans (Table A-VII). The best design (namely A) results in being the most cost-effective. Although it has a performance below the nominal requirements, it will support the number of terminals nominally required. The selected design is a trade-off among technical and economical characteristics, a trade-off which is best in relation to the selection plan specifications.

# Extracting unique rows of a bounded degree array using tries

by DOUGLAS COMER

Purdue University  
West Lafayette, Indiana

## SUMMARY

An array with integer entries between 0 and  $d-1$  has bounded degree  $d$ . This paper considers several algorithms for extracting the set of unique rows from a bounded degree array. For each algorithm considered, it gives the time, space, and I/O requirements, and an assessment of the types of applications for which the method is well-suited. It begins with four methods based on well known techniques and data structures, and goes on to propose a new algorithm which uses a form of digital search tree known as a trie. It shows that the trie-based scheme has advantages over the other methods. Finally, it discusses some applications including computing a projection in a relational database system, and finding classes of isomorphic rows.

## INTRODUCTION

A 2-dimensional array  $A$  is of *bounded degree*  $d$  if each element of  $A$  is an integer,  $a$ , such that  $0 \leq a < d$ . The problem of extracting the unique rows of an array with bounded degree arises in many applications. The UNIX operating system [7] provides the command "uniq" which extracts all unique lines from a text file. In [5], Housel develops an algorithm for scheduling processes in a data restructuring program. One step of the algorithm finds all unique rows in a bounded degree matrix.

Finally, in a relational database system [2], each *relation* may be thought of as a 2-dimensional array in which each column is a *domain*. The process of *projecting* a relation over a subset,  $P$ , of its domains consists of eliminating all domains (columns) not in  $P$  and extracting unique rows from the resulting subarray. For a relation,  $R$ , with domains "employee name" and "county of residence," a projection of  $R$  over "county of residence" would be a list of all those counties in which employees reside. Since the cost of removing a subset of columns from a given row is usually small, the difficulty in computing such a projection is essentially that of extracting the unique rows from an array.

In order to quantify the cost of various solutions to the

problem of extracting unique rows, we state it as follows:

*Problem 1 (row compression):* Let  $A$  be an  $n \times m$  array of bounded degree  $d$ , and let  $k$  be the number of distinct rows of  $A$ . Find a  $k \times m$  array,  $A'$ , such that each row of  $A$  appears in  $A'$ .  $k$  may not be known *a priori*.

Since we are interested in the space required to generate  $A'$  as well as the time required, we will assume that  $A$  is stored on secondary storage by row and need not be kept in main memory. We further assume that  $A$  is "unordered" in the sense that the rows are not arranged lexicographically (the ordering of  $A'$  is discussed below).

Several solutions for Problem 1 are reviewed below which are based on well known algorithms. Knuth [6] is a good reference for both the detail and analysis of the sorting and hashing algorithms mentioned.

Solutions to Problem 1:

- 1) Insertion: For each row,  $r$ , of  $A$ , if  $r$  is not in  $A'$  insert it.
- 2) Comparative Sort: Read  $A$  into memory and sort it using a comparative sort (like quicksort), placing the rows in lexicographical order.
- 3) Radix Sort: Proceed as in #2 using a radix sort.
- 4) Hashing: Hash the rows of  $A$  into a table, skipping duplicates and adding the rest to  $A'$ .

Each of these solutions may have advantages over the others depending on  $n$ ,  $m$ , and  $k$ . Method 1, insertion, is easy to program, requires only space for  $A'$ , and reads  $A$  in row order. It requires only  $nm/b$  disk fetches, where  $b$  is the blocking factor. To compare two rows takes  $m$  comparisons, so if  $A'$  is kept ordered and a binary search is used, the running time is  $O(mk^2 + nm \log_2 k)$ , where the term  $mk^2$  accounts for inserting a row in order. By using  $k$  extra locations for pointers and not actually moving the rows of  $A'$ , the time can be reduced to  $(k^2 + nm \log_2 k)$ . For small  $k$ , the running

time is reasonable even if  $n$  is large. But if  $k$  is as large as  $n$ , the running time is  $O(n^2 + nm \log_2 n)$ . We shall see that other methods have much better running time.

Method 2, a comparative sort, is practical when  $k=n$  (i.e., there are only a few duplicate rows in the array), and  $n$  is large. The running time is only  $O(nm \log_2 n)$  and the space required is  $nm$  plus locations for  $n$  pointers (to eliminate moving rows of  $A$ ). Since the array has bounded degree, an immediate improvement in running time to  $O(nm)$  can be obtained by using Method 3, a radix sort, with radix  $d$ . Both of these sorting methods also have the advantage that the rows of  $A'$  can be generated in either sorted order or in the original order.

Since radix sorting requires time proportional to the size of the input, no faster method can be found in the general case. However, processing in the input in column order, as is done in a radix sort, requires  $nm$  disk fetches for reasonable size array  $A$ , while processing the input by row requires only  $nm/b$  disk accesses. The number of entries per block,  $b$ , would normally be high, making radix sort unreasonable. In such cases Method 4, hashing, would be desirable since it processes  $A$  by row in one pass while still using only  $O(nm)$  steps. But there are drawbacks to hashing as well. First, the new array,  $A'$ , can no longer be output in sorted order without a separate procedure. Second, since the number of distinct rows of  $A$  is not known *a priori*, the hash table may be allocated (and initialized) much larger than necessary.

We seek a solution to Problem 1 which meets the following criteria:

1. No more than  $O(nm)$  steps are required,
2. No more than  $O(km)$  space is taken (where  $k$  is not known *a priori*),
3. The new array,  $A'$ , can be generated in either sorted order or in the original order, and
4. The array  $A$  is processed by row in a single pass, requiring only  $nm/b$  disk accesses.

## A TRIE-BASED METHOD

In this section a solution for Problem 1 which meets the four criteria listed above is presented. Like the other methods discussed, this one is based on a well known idea, that of a trie index. The definition of a trie will be given first, and then its use in solving Problem 1 will be discussed.

Tries were introduced by de la Briandais [3] and Fredkin [4] for the storage of character data. Sussenguth [9] proposes an alternative implementation which requires more time to access but saves space. In this paper we will give a slightly modified definition of a trie and relate a trie to an array of bounded degree.

*Definition:* Let  $A'$  be a  $k \times m$  array of bounded degree  $d$  such that row  $i \neq$  row  $j$ ,  $i \neq j$ . A *trie* for  $A'$  is a tree with  $k$  leaves, each of which lies at depth\*  $m$  such that:

\* the root of a trie lies at depth 0; the children of a node at depth  $i$  lie at depth  $i+1$ .

1. For each row of  $A'$  there is a path in the trie from the root to a leaf with the sequence of labels on edges in the path equal to the sequence of elements of the row, and
2. Each such path in the trie has a sequence of labels on the edges equal to a row of  $A'$ .

Figure 1 shows an array and the trie for it.

To search for a row in the trie, one begins at the root and follows those edges with labels which are the same as the elements of the row in question. An important property of tries is that the decision about which edge to follow at a given node can be made in constant time. Fredkin's implementation uses an array of pointers at each node to achieve the property. To follow an edge with label  $p$ , one follows the  $p^{\text{th}}$  pointer. Of course, the range of label values determines the storage necessary at each node. For a trie corresponding to an array with bounded degree  $d$ , each node would have  $d$  pointers.

Since the decision about which edge to follow takes constant time, searching for a row of length  $m$  requires  $O(m)$  steps. Adding a row to an array corresponds to adding a leaf to the trie and establishing a path from the root to the leaf. Knuth [6] provides a detailed algorithm for insertion and shows that it will require only  $O(m)$  steps. Thus, to build a trie for a  $k \times m$  array will require  $O(km)$  steps.

We now address the solution of Problem 1 using a trie. The method is straightforward: for each row,  $r$ , of  $A$ , if  $r$  is not in the trie, insert it. Since the search and insertion algorithms are nearly identical, they can be merged. A search continues until a null pointer is found, at which time the addition of a new path begins. Therefore, any row can be processed in  $O(m)$  steps, so  $A$  can be processed in  $O(nm)$  steps, the minimum possible. The constant overhead is very small, making the method good in practice. Furthermore, if storage is allocated on demand, only  $O(km)$  space will be used for the trie (even though  $k$  is not known before the trie is begun).

Consider the four criteria for a row compression algorithm outlined in the previous section. We have already shown that a trie is as fast as possible, and uses only  $O(km)$  space.

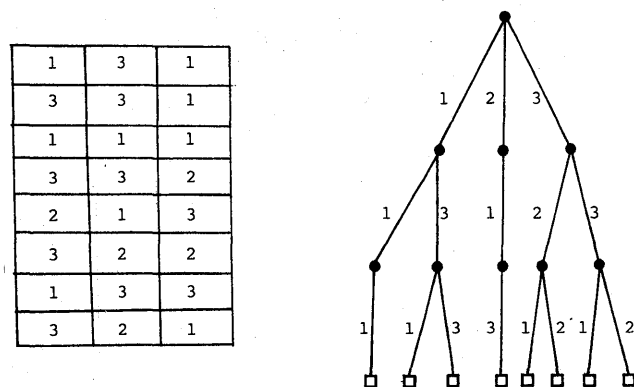


Figure 1—An array of degree 3 and the trie for it.

To see that a trie can be used to order  $A'$ , observe that a preorder traversal of the trie (details of a traversal can be found in [8]) will generate  $A'$  in sorted order. To obtain  $A'$  in the original order, the trie can be constructed while  $A'$  is being generated: new rows are output as they are inserted into the trie, while duplicate rows are ignored. Since  $A$  is processed by row, the trie-based method needs only  $nm/b$  disk accesses while reading  $A$ . Therefore, the method meets the criteria listed above.

Notice that the trie-based method does well in the application of computing projections because the extraneous columns do not actually have to be removed before the trie is built. Instead, one can read a row of the file representing the relation, and pick out exactly those columns that should remain in the relation as the trie is searched.

## OTHER APPLICATIONS

One way to look at the problem of extracting the unique rows of an array is to think of placing all equal rows in the same class. In some applications, one would like to group together all those rows which are isomorphic (equal up to a renaming of values), and extract one representative of each such class. For a binary matrix, a trie can be used to find classes of isomorphic rows using a minor variation of the method outlined above. As each row is inserted into the trie, the first element is examined. If it is a 0, the remaining elements in the row are inserted into the trie as usual. If the first element is a 1, the complement of the row is inserted. Each leaf in the trie is the head of a list of the row numbers of all rows which terminate there and represents a class of rows which are isomorphic. By chaining the leaves together as they are added to the trie, a list of the classes of isomorphic rows can be obtained in  $O(nm)$  time for an array of  $n$  rows and  $m$  columns.

For arrays with degree greater than 2, the process of finding classes of isomorphic rows becomes more complicated. The trick is to scan the row once, changing entries to a canonical form before the row is inserted into the trie. As the row is scanned, each entry is examined to determine whether that value had been seen before. If it had, then it is mapped into the same integer as it was before. Otherwise, a fresh integer is assigned as its code, the entry is changed into the new integer, and the integer is stored for use with later entries in the row with the same value.

An unsophisticated algorithm for mapping a row of  $m$  entries into canonical form follows. In the algorithm,  $r$  is the row vector,  $c$  a counter, and  $v$  a vector of length  $d$ .

```

c := -1;
for i := 1 to d do
  v[i] := -1;
for i := 1 to m do begin
  if v[r[i]] = -1 then begin
    c := c + 1;
    v[r[i]] := c
  end; (* if *)
  r[i] := v[r[i]]
end; (* for *)

```

This implementation, which requires  $O(d)$  overhead per row as well as space for a vector of length  $d$ , is quite practical for small  $d$ . In fact, the cost of building a trie will usually dominate the small overhead incurred in changing the rows into canonical form.

A more sophisticated approach makes use of the "constant-time array initialization" mentioned in [1]. This result shows that the vector  $v$  could be initialized in constant time provided some extra space is used. The total space used is roughly  $3d$  which makes the method useful for even moderately large  $d$ .

## CONCLUSIONS

We have shown that a trie index can be an efficient method for extracting unique rows from an array of bounded degree. The method is fast, uses storage only as necessary, processes the input array by row, and can be used to generate the output in sorted order.

One application of the trie-based method is that of computing the projection of a relation in a relational database system. Other applications include extracting classes of isomorphic rows in a bounded degree array.

## REFERENCES

1. Aho, A., Hopcroft, J., and Ullman, J., *The Design and Analysis of Computer Algorithms*, Addison Wesley, 1974.
2. Date, C., *An Introduction to Database Systems*, Addison Wesley, 1975.
3. de la Briandais, R., "File Searching Using Variable Length Keys," *Proc. Western Joint Computer Conference*, 1959, 295-298.
4. Fredkin, E., "Trie Memory," *CACM* 3:9 (Sept. 1960), 490-499.
5. Housel, B., "A Technique for Implementing Data Restructurers," *TODS* (to appear).
6. Knuth, D., *The Art of Computer Programming, Vol. 3, Sorting and Searching*, Addison Wesley, 1973.
7. Ritchie, D. and Thompson, K., "The UNIX Time-Sharing System," *CACM* 17:7 (July 1974), 365-375.
8. Reingold, E., Nievergelt, J., and Deo, N., *Combinatorial Algorithms: Theory and Practice*, Prentice Hall, 1977.
9. Sussenguth, E., "Use of Tree Structures for Processing Files," *CACM* 6:5 (May 1963), 272-279.



# A look at making the ADP procurement process more efficient—Temporary regulation 46

by ROGER J. GORG, GEORGE N. BAIRD and JUDITH A. PARKS

*Federal Compiler Testing Center*  
Falls Church, Virginia

## INTRODUCTORY NOTE

This paper represents the view of the authors and in no way is to be taken as an official Government or General Services Administration position on the subject matter. It represents the authors' experience in acquiring a computer system under existing Federal Procurement Regulations.

## BACKGROUND

The Brooks Act (Public Law 89-306)<sup>1</sup> went into effect on October 30, 1965. The purpose of the act was to provide for the economic and efficient purchase, lease, maintenance, operation and utilization of Automatic Data Processing (ADP) Equipment by Federal departments and agencies. The Administrator, General Services Administration was given the authority and direction to provide the necessary mechanisms for the economic and efficient purchase, lease and maintenance of ADP Equipment by the Federal Government. As a result, GSA assumed the procurement authority for all general purpose computer systems to be acquired for Federal use. Appropriate Regulations regarding the acquisition of ADP Equipment were prepared and issued.

The procurement regulations developed and implemented for the purchase of ADP equipment put into effect at that time required all Federal Agencies to request a Delegation of Procurement Authority (DPA) from GSA for any acquisition over \$50,000. This meant that an agency procuring ADP equipment with a cost greater than \$50,000 had to include GSA review and approval in the procurement process. At the time the regulation went into effect few computer systems were being purchased for under \$50,000. The net effect was that GSA had to expend some of its limited resources whenever a computer system was required by any agency in the Federal Government, regardless of the cost or size of the system.

Federal Agencies and the ADP industry in general also had had a growing concern about the lengthy, complex and costly ADP procurement process. Much of this concern had been directed at the disproportionate cost to participate in and the large amount of time involved for a procurement to be completed. This issue was of particular concern in the

case for which the dollar value was under \$1,000,000. According to government studies, the average cost for the Federal Government to conduct a competitive procurement in this dollar range was about \$12,000 in administrative costs. The average time to complete such a procurement was about 6 months.

The acquisition procurement of major computer systems by means of a traditional functional specifications ADP procurement was generally more time consuming and administratively costly than a non-competitive (sole-source) or equipment specification procurement. However, the potential benefits which could be realized through a fully competitive procurement would, in the long run, far outweigh the short term disadvantage. GSA was working on regulations which would speed up the process and reduce the cost of conducting procurements for low cost ADP equipment where the benefits of a traditional procurement were not as great.

In October 1976, the House Committee on Government Operations (the Brooks Committee) issued a report to Congress on the administration of the Brooks Act covering the procurement of ADP resources by the Federal Government.<sup>2</sup> The report pointed out several areas where GSA, the Office of Management and Budget (OMB) and the National Bureau of Standards (NBS) had not administered and implemented the Brooks Act in accordance with the original wishes of Congress. The report also noted that 56 percent of procurement delegations in 1975 were for procurements under \$250,000. These procurements mainly involved minicomputers, peripherals, software and maintenance where a highly competitive market exists. Agencies were in effect being required to follow the same procedures for small-dollar buys as they were for the purchase of major mainframe computer systems. GSA determined that the dollar value of these smaller procurements represented only about 6 percent of the total value of all ADP equipment procurements.<sup>3</sup>

The Congressional report recommended that, in order to conserve the resources of both GSA and user agencies, new procedures had to be adopted which would allow agencies to procure small-dollar systems without having to obtain a delegation of procurement authority. While this approach was intended to speed up the procurement process and save the government time and money, it was in no way designed



to erode or eliminate the competitive aspect of the procurement.

The most viable solution to making the needed changes involved the potential use of the small purchase procedures (which previously were not available for use with ADP equipment procurements) and a revision to the criteria for the use of the ADP schedule contracts issued by GSA's Automated Data and Telecommunications Service. The expanded use of the schedule contracts would allow purchase orders to be placed expeditiously and at the least possible expense to both the Government and industry.

(Note: ADTS/ADP schedule contracts represent pre-negotiated prices, terms and conditions under which the ADP Equipment manufacturers offer their products to the government. These contracts are negotiated on a fiscal year basis and establish the price for each of the items listed. A maximum order limitation (MOL) is specified in terms of a number of systems and/or a dollar value for each order to be placed. This limitation can be waived by GSA. Traditional ADP competitive procurements based on functional-type specifications have resulted in the participating vendors offering the government prices below those shown in their schedule contracts.)

#### TEMPORARY REGULATION 46

GSA implemented the recommendation in the Brooks Committee report by issuing Temporary Regulation 46<sup>3</sup> entitled "Use of Small Purchase Procedures and Schedule Contracts for Automatic Data Processing (ADP) Resources." This temporary regulation prescribed the use of small purchase procedures for ADP procurements of up to \$10,000, revised agency procurement procedures for competitive procurements which do not exceed a purchase cost of \$300,000 and provided policies and procedures for the use of schedule contracts for ADP Equipment.

This regulation does not suggest or imply a non-competitive or sole-source procurement, but requires that maximum practicable competition be preserved and clearly documented by the procuring agency. All procurement statutes, policies and regulations that apply to a traditional ADP functional specification procurement also apply to a Temporary Regulation 46 (schedule contracts) procurement.

A procurement of this type would take place as follows. First, the agency must have the necessary bona fide need, funding and authority to initiate the procurement. Then the agency's requirements for the computer system must be defined and documented. Next, the literature and pricing guides of the computer systems available through the ADP schedule contracts must be reviewed and analyzed in terms of agency requirements and cost. Contact with and presentations given by the schedule contractors can be useful in performing this analysis. Included in this phase must be the accurate documentation of the actual technical and costing analysis performed and the identification of the system that best meets the agency's needs in terms of cost, price, and other factors considered. (The total cost of the system is based on schedule purchase price regardless of method

of acquisition. The cost may not exceed \$300,000 under this procurement procedure. Annual maintenance is not included in calculating the total cost.) If the total cost of the system is under \$35,000, the system can be ordered off the appropriate ADP schedule contract immediately. If the total system cost is greater than \$35,000, then the government must publicize its intention to order the system by placing a synopsis in the Commerce Business Daily (CBD) in advance of placing the order. This permits interested vendors, who may not be available under the schedule contracts program, to demonstrate their ability to satisfy the agency's requirements. If no responses are received by the deadline given in the synopsis, the procurement file is so documented and the order is placed with the selected schedule vendor.

If any responses are received from vendors who have not already been fully considered, the procuring agency must be able to document that the use of the ADP schedule contract will be most advantageous to the agency/government. (If that position cannot be demonstrated, the agency must revert to a traditional ADP competitive procurement.) Should the agency choose to select a system at other than the lowest price available under any GSA schedule contract, then the agency must justify its action.

The remainder of this paper discusses two Temporary Regulation 46 procurements within GSA, the results and an assessment of the success of these procurements.

#### FEDERAL COMPILER TESTING CENTER — MINI COMPUTER ACQUISITION

The Federal Compiler Testing Center (FCTC) (formerly known as the Federal COBOL Compiler Testing Service) was transferred from the Navy to the General Services Administration (GSA) in May 1979. The computer support provided by the Navy prior to the transfer was guaranteed until the end of the fiscal year. Part of the transition plans of the organization required the acquisition of an in-house computer system to handle the majority of the FCTC data processing workload. Since the estimated cost of a system was around \$275,000, the procedures described in Temporary Regulation 46 established the applicable procurement vehicle by which the computer system should be purchased most efficiently.

The requirements for the computer system had to be defined in order to determine which computer systems available under the ADP schedule contracts could meet those needs. The requirements were developed in the form of a set of functional-type specifications. There were several reasons for this. First, the FCTC staff was accustomed to expressing technical requirements using this method of representation. Second, the FCTC needed a document for use in determining which vendors under the ADP schedule contracts met our minimum software/hardware requirements. Third, technical requirement had to be established for comparison and evaluation purposes. And finally, should the FCTC be placed in a position where a traditional ADP competitive procurement had to be initiated, the technical specifications would be ready.

After the system requirements were defined, the schedule vendors with equipment which appeared to meet the minimum requirements were contacted and technical literature was requested.

One major software requirement that made it easy to identify which computer systems would be potentially acceptable was the requirements for an identifiable Low-Intermediate level of Federal Standard COBOL (COBOL 74)<sup>4</sup> with the stipulation that the COBOL compiler had to have been officially validated at the time the system was purchased. This requirement was extremely critical since one of the major functions of the FCTC is to produce and maintain Compiler Validation Systems for COBOL.<sup>5</sup> A second standard programming language, FORTRAN 78<sup>6</sup> (X3.9-1978 American National Standard Programming Language FORTRAN), was also necessary because the Center will be producing a FORTRAN Compiler Validation System for the FORTRAN standard for use beginning in calendar year '81.

Next, technical literature from each vendor which satisfied the basic programming language requirements was examined to determine if the rest of the mandatory requirements were met. Of the seven systems which met the programming language requirements that were being reviewed, several did not meet other mandatory requirements. For example, one system could provide a 9 Track 800 BPI magnetic tape drive but did not support a magnetic tape drive capable of handling 1600 BPI. Since 1600 BPI is the standard recording density used by the FCTC for distributing software, this deficiency would have created a time consuming and expensive situation if the in-house system could not be used to perform the simple task of copying magnetic tapes at the required 1600 BPI density.

At the conclusion of the elimination procedure utilized, five systems met all mandatory requirements. At this point, the ADP schedule contract price lists provided by each of the vendors were examined to determine the lowest cost system. This was not a simple task since there was generally insufficient information contained in the price schedules to determine how to configure a system and develop the associated cost for that system. Although the key items and associated prices could be identified, it was difficult to determine what devices were compatible with the CPU, the number of devices which a single controller could support, or whether all the necessary components and cabinets had been identified. To order a computer only to find out it could not be used without the purchase of additional equipment would be totally unacceptable and improper.

The vendors under consideration were contacted and asked to provide a presentation on their systems and answer questions regarding the configuring of each system. This process provided the final information necessary to complete the evaluations of the computer systems being considered. Shortly thereafter, the final analysis was performed and the most cost effective (least expensive) system that met the mandatory requirements had been identified. The official GSA prices under the ADP schedule provided by the vendors had to be closely monitored during the evaluation process. One vendor lowered his schedule prices requiring a recalculation of the total cost for that system.

Once the analysis was complete, a synopsis for the Commerce Business Daily was prepared and, along with the results of the analysis, was forwarded to the Contracting Officer in charge of the procurement. The synopsis contained a functional description of the computer system that had been selected. It was described in terms of the number of concurrent users that must be supported, the amount of mass storage space that was required, the number of magnetic tape drives required, etc. However, the Contracting Officer felt that the synopsis should reflect the exact system configuration that was to be ordered, rather than a general description. The synopsis which was developed on that basis included a description of the system and a notice of the intent to order the system from the selected ADP schedule contract and was published in the CBD. The synopsis gave public notice that eight days were being allowed for interested vendors to respond and demonstrate that they meet the requirements specified and could provide the equipment necessary to perform the job at a lower overall cost to the Government.

During the eight day response period, only one response was received from industry. The system offered in the response did not meet all of the mandatory requirements and was roughly 60 percent more expensive than the selected system. Once the final analysis and appropriate documentation of this response was prepared and provided to the Contracting Officer, the official procurement action was initiated the following day. Delivery was to take place within the next 90 days which is the standard time required under the ADP schedule contracts.

#### *Assessment*

The procurement was successful and completed in a little over three months as compared with a normal timeframe of up to 18 months to acquire an ADP system without employing the provisions of FPR 46. It took roughly five weeks from the time the Contracting Officer was assigned to the project until the delivery order was signed. Two of these five weeks were required because the initial synopsis that appeared in the Commerce Business Daily (CBD) allowed only one working day for responses (the time for the synopsis to be received and published by the Commerce Business Daily had been under-estimated) and the synopsis had been somewhat garbled when rewritten in the Contracting Office before being mailed to the CBD. So a net time of three weeks time was required from the time the Contracting Officer was first involved. Approximately 50 hours were spent by the Contracting Officer in this procurement, several orders of magnitude less than would have been required using the traditional ADP procurement approach.

The time required to prepare the necessary justifications and the technical specifications was just under 240 hours spread over six weeks using the resources of four technical people. It is critical in this type of procurement that the time necessary for planning and defining the requirements for the acquisition are not reduced or else the result may be a genuine catastrophe. The review of the technical literature and the presentations provided by the vendors were extremely

useful since under Temporary Regulation 46 there is no benchmark or functional demonstration involved. This meant that the technical literature and vendors' presentations had to be used to determine if the system could handle the Center's workload. If the proper configuration cannot be determined, then it would be better to revert to the traditional ADP procurement which includes benchmarking and functional demonstrations rather than select a system that may not be adequate to meet the requirements of the agency. This review took about 150 hours over a three week period.

The problem areas in this procurement were few and easily identifiable. First, the Contracting Officer had not performed a Temporary Regulation 46 procurement before. Therefore, a learning process was involved. Temporary Regulation 46 was reviewed several times during the acquisition process and several detailed discussions were held with the Contracting Officer regarding the exact way to proceed at each step of the acquisition. A good part of the time the Contracting Officer was operating under the misapprehension that the procurement was sole-source and treated the procurement as such. Accordingly, it was not until late in the procurement process that the Temporary Regulation 46 procedures took hold. A similar type of problem with Contracting Officers not being familiar with Temporary Regulation 46 was also mentioned at the "Federal ADP Procurement" course (presented by AIIE, September 25-27, 1979 in Washington, D.C.) by several speakers and members of the audience. It appears that the process is so simple compared to other ADP procurement processes that contracting officers find it hard to believe that this new procedure is legal and become skeptical. However, this problem was overcome.

The second problem had to do with the price lists provided by the vendors under the ADP schedule contracts program. It was next to impossible to configure and/or price a complete system with the information provided. An improvement to the schedule price lists would be the inclusion of a configuration guide to help the reader understand what is necessary to combine hardware devices and configure a usable system. Some vendors' literature was easier to use than others, but none were sufficient by themselves and help was required from the respective vendors to understand how to configure a system.

#### NATIONAL AUDIOVISUAL CENTER—ADP EQUIPMENT ACQUISITION

The second Temporary Regulation 46 procurement conducted by GSA involved the acquisition of ADP equipment for the National Audiovisual Center. While many similarities existed between this procurement and the one previously described, certain experiences were encountered which offer additional insight into the procurement process for small systems.

The National Audiovisual Center (NAC) required equipment to accommodate the processing of their audiovisual information and distribution systems. Requirements included both the conversion of existing systems and the de-

velopment of new systems for implementation on new equipment. Documentation was available on existing systems and the new systems were described in detailed ADP design packages. Using this workload information, a requirements document was developed. The document contained a brief description of the workload, system interfaces and hardware/software requirements. Additionally, to meet the COBOL programming and conversion needs of the National Audiovisual Center, a requirement was included to the effect that any COBOL compiler considered for acquisition must have at a minimum been validated at the Low-Intermediate level of Federal Standard COBOL. (This requirement is consistent with Federal Property Management Regulation 101-36.1305-1 "Federal Standard COBOL" which requires that one of the four levels of Federal Standard COBOL be identified for any COBOL compiler that is to be brought into the Federal inventory.)

Upon completion of the requirements definition task, technical specifications were researched to determine candidate vendors. Eight vendors were identified who offered products of comparable size and capability which appeared to be able to satisfy the requirements. A more detailed analysis of the various system capabilities and the specific requirements (e.g. COBOL validation, serving as both a host system and a RJE system, etc.) revealed that five of these vendors failed to meet certain of the mandatory requirements. For the remaining three systems which had been identified, configuration charts were prepared. These charts were subsequently validated to ensure their accuracy. Cost comparisons were performed using ADP schedule prices and a recommended selection was made.

The contracting officer was apprised of the procurement selection rationale. He was provided a configuration list of the selected system, and a synopsis for publication in the Commerce Business Daily (CBD). The synopsis was published and fifteen days were allowed for responses. Unlike the announcement in the CBD for the Federal Compiler Testing Center (FCTC) procurement, this notice did not identify the selected vendor or provide a detailed configuration. As a result, inquiries were received from vendors requesting complete bidding information. All interested parties were sent supplemental information which included a detailed configuration showing actual model numbers. They were informed that the evaluation of all known schedule sources had been completed and a determination made that this specific configuration met all Government requirements. They were further told that if they could provide a system with an equivalent configuration and appropriate schedule price references, it would be fully considered. The reason "appropriate schedule price references" were requested was that we had found in our earlier analysis that the most recent ADP schedule prices were not always listed in the published documents. As mentioned in the FCTC procurement above, it was possible for schedule prices to have been lowered significantly during the fiscal year.

In total eleven vendors indicated an interest in responding to the notice. Only four actually submitted price quotations, one of which was invalid because it was received after the fifteen day lapse time. The three valid price quotations, in-

cluding one submitted by the selected offeror, was evaluated initially on price. The result was that all prices fell within a 10 percent range and the system which had been selected still represented the lowest cost to the Government. Therefore, only a cursory technical review was made of the other proposals. A recommendation was presented to the contracting office that a purchase order be issued for the selected system. This final step was accomplished in accordance with the recommendations provided.

### Assessment

Several observations can be made in respect to this procurement. The resources required to conduct the procurement approximated those described for the FCTC procurement, with the exception that the contracting spent in the neighborhood of 22 hours. One significant difference from the FCTC procurement was the close competitive range of ADP Schedule price quotations for equivalent configurations. This situation leads us to believe that when new schedules are negotiated, in light of Temporary Regulation 46, the Government should be in a good bargaining position and vendors should be receptive to lowering prices for competitive reasons. Our experience indicates that a complete analysis of user requirements and the resulting hardware/software needs is important and well worth the time and effort required. Like the Testing Center participants, we also found that configuring from ADP schedules and accompanying literature is a difficult job. Both industry and the Government should strive for more standardized formats and descriptive narrative in the ADP schedule contracts in order to facilitate their use under Temporary Regulation 46.

### SUMMARY

Based on the two procurements described above, the new procurement process established for limited dollar level ADP equipment appears to be a good one. The calendar time involved in the selection of this system was only around three months including the definition of the system requirements. The procurements, in terms of the time required from when the Contracting Officer became involved until the computer

systems were ordered was a relatively short time—five weeks in the case of the FCTC which could have been three weeks except for the need to re-issue the CBD notice and four weeks for the NAC which would have been shortened to three weeks had they allowed only one week for comments on the synopsis in the CBD.

The approximately 400 hours of professional time (not including Contracting Officer time) that was spent by both the FCTC and NAC would have been at least three times that amount if a benchmark and live test demonstrations had been required for a traditional ADP equipment procurement. The time and expense of the vendors was limited to a few phone calls, providing copies of some of their technical literature, and several meetings. This certainly represents far less cost than had they participated in a traditional ADP procurement.

Although the two Contracting Officers approached the procurements differently, the results were closely the same. The problems associated with a new procurement process (Temporary Regulation 46) will require the time necessary for government Contracting Officers to become familiar and comfortable with the new process. It appears that when dealing with the synopsis to be published in the Commerce Business Daily, a description of the system (which has been selected) rather than a functional description is desirable. This method of announcement should keep down responses from vendors who were already evaluated and rejected early in the procurement. It would save the vendors time and money as well as avoiding unnecessary duplication of effort by the government which must review all responses regardless of previous determinations in the selection process.

### REFERENCES

1. Public Law 89-306, 89th Congress, H.R4845, 1965.
2. House Report No. 94-1746, "Administration of Public Law 89-306, Procurement of ADP Resources by the Federal Government," 1976.
3. Title 41, Chapter 1, Federal Procurement Regulation, FPR Temporary Regulation 46, "Use of Small Purchase Procedures and Schedule Contracts for Automation Data Processing (ADP) Requirements," *Federal Register*, Vol. 43, September 28, 1978.
4. "American National Standard Programming Language COBOL, X3.23-1974," American National Standards Institute, New York, 1974.
5. Federal Property Management Regulation 101-36.1305-1 "Federal Standard COBOL," *Federal Register*, Vol. 44, August 28, 1979.
6. "American National Standard Programming Language FORTRAN, X3.9-1978," American National Standards Institute, New York, 1978.



# An information base for procedure independent design of information systems

by LEVENT ORMANCIOGLU

University of Wisconsin-Milwaukee  
Milwaukee, Wisconsin

## PROCEDURE INDEPENDENT DESIGN

An information system contains a vast number of logical statements in addition to data. These logical statements are used to control the retrieval and use of data for applications. Systems Design is the process of translating the information requirements of an organization to a set of logical statements which operate on the observed data to generate new data. The 'derived data' generated in this fashion could actually be used for decision support or stored back in the data base for future use; however, the storage is rare since it can always be regenerated using the logical statements. The logical statements are often expressed using a procedural programming language mostly aided by Data Base Management Systems which accomplish to decouple the physical data structure and the application system designer's logical view of it. This feature, called 'data independence,' has a number of advantages such as:

- a. simpler interface with the applications programmer
- b. machine optimized physical design
- c. modifications in the physical structure without changing the application programs
- d. high portability to different computer systems.

The main difficulty with data independence is the fact that shielding the application programmer from the physical data structure also deprives him of the information needed to develop efficient algorithms. To restore efficiency without sacrificing data independence requires delegating the responsibility of generating algorithms to the information system. This responsibility includes not only the generation of data retrieval algorithms but also application system algorithms since the efficiency of both are strongly interrelated. The application system designer in such an environment would have to avoid specifying algorithms and procedures; and restrict himself only to specifying the functional relationships between the data to be generated and the data in storage. This approach to design is called 'procedure independent' and Information Base is an approach based on Predicate Calculus and Array Algebra to the procedure independent design of information systems.

A number of data sublanguages based on Predicate Cal-

culus have been developed including ALPHA,<sup>1</sup> COLARD,<sup>2</sup> and RIL;<sup>3</sup> and a number of others with more user orientation like QUEL,<sup>4</sup> SQUARE,<sup>5</sup> and QUERY by EXAMPLE<sup>6</sup> are based on similar principles. All of them proved to be relationally complete which refers to their selective ability in terms of identifying the subsets of data. However, in addition to data selection, an information system needs the ability to algebraically manipulate the data to generate new data. This need has long been recognized as a need to host a data selection sublanguage with a programming language. Information Base uses APL operators<sup>7</sup> to algebraically manipulate data which has been retrieved using predicate calculus. The capability of iteration between algebra and calculus increases the power of the language.

## INFORMATION BASE

The development of the Relational Data Base Theory<sup>8</sup> and the use of first order predicate calculus made it possible to view data as a collection of two dimensional tables and retrieve subsets of data non-procedurally. An information base adds to these capabilities in two ways:

- a. by developing a 'function specification language' (FSL) based on predicate calculus and array algebra to express the functional relationships between the data to be generated for applications and the data in storage;
- b. by storing the functional expressions in FSL—called functions—in a 'function base' and identifying each function by its unique output. This makes it possible to treat the functions as a set without reference to their sequence.

FSL is an all-purpose language used to specify functions, to define and manipulate data and to enter queries. A query expressed in FSL goes through three stages and a number of iterations among them to produce a response:

- a. the query is interpreted as a function to determine its input data requirements;
- b. for each data item required but not stored explicitly, the appropriate function is retrieved from the function

base to generate the data. The search strategies are the same as data bases, since the function base is treated as a set;

- c. for each function retrieved above, the data requirements are determined by interpretation as in (a) and the process is repeated.

The iteration between functions and data continues until the query is expressed only in terms of explicitly stored data. The process basically consists of then, repetitive substitution of 'implicit data' with corresponding functions until a query as an expression in FSL is defined in terms of the explicit data only. At that stage the expression is executed to produce a response to the query.

Example: A query involves the computation of discount which is 10 percent of the total cost of each supply. The explicit data contains supply\_quantity and the supply\_price. A function in the base states that:

$$\text{supply\_cost} \leftarrow \text{supply\_quantity} \times \text{supply\_price}$$

The query should state:

$$\text{supply\_discount} \leftarrow 0.1 \times \text{supply\_cost}$$

where supply\_cost is substituted by the system with supply\_quantity  $\times$  supply\_price; and the final expression is executed to produce the response.

## FUNCTION SPECIFICATION LANGUAGE (FSL)

First order predicate calculus has been shown to be a convenient and powerful data selection sublanguage<sup>9</sup> but not an algebraic manipulation language for applications. The algebraic manipulation of retrieved data is usually done either through a programming language which adds undesirable procedure to the design or by adding built-in functions like SUM, ADD, AVERAGE which usually produce a less than complete or an inflexible language. FSL augments predicate calculus with array algebra utilizing APL operators. Array algebra requires positional correspondence between elements of different arrays and hence cannot be applied directly to relational tables where the order of tuples (rows) is arbitrary. The strategy in this paper is to use a two-step process where predicate calculus is used to determine the relevant subsets of data and to feed them into vectors of an APL expression before execution.

FSL syntactically allows any expression  $S$  where:

- $S = P.t$  where  $P$  is a relation name and  $t$  is a domain name;
- $S =$  any valid APL expression (without branching) involving  $S$ ;
- $S = S$ : any predicate calculus expression involving  $S$ .

The basic unit of information denoted by  $P.t$  refers to domain  $t$  of relation  $P$  and interpreted as a set by predicate calculus expressions and as a vector by APL expressions.

The basic FSL statement involves an APL expression augmented by a predicate calculus expression. Given an APL expression involving  $n$  vectors of type  $P.t$ , the predicate calculus expression augmenting it is interpreted first to produce a response set of  $n$ -tuples. These  $n$ -tuples are then fed into  $n$  vectors of the APL expression for execution.

Predicate calculus expressions consist of well formed formulas (wff) defined as follows:

- a. Atomic formulas containing constants, variables in the form  $P.t$  where  $P$  is a relation name and  $t$  is a domain name, and arithmetic comparison operators  $=, \neq, <, \leq, >, \geq$  are wff.
- b. If  $A$  and  $B$  are wff, then:  
 $\sim A, A \vee B, A \wedge B, A \rightarrow B$  are wff.
- c. If  $A$  is a wff and  $P$  is a relation name:  
 $\forall P(A), \exists P(A)$  are wff.
- d. The formulas obtainable by finitely many applications of  $a, b,$  and  $c$  are wff.

Relation names are used as variable names with quantifiers  $\forall$  and  $\exists$ —as indicated in c—to improve readability. If more than one variable corresponds to a particular relation  $P$ , primed variables  $P', P''$ , etc. are used to distinguish variables. Comparison operators are assumed to have precedence over logical operators.

A list of APL operators used in FSL are given in the Appendix. Most of the examples will be restricted to arithmetic operators  $+, -, \times, \div$ ; maximum  $\Gamma$ ; minimum  $L$ ; size  $\rho$ ; concatenation  $;$ ; and reduction  $/$ ; where the last one is used to reduce a vector into a scalar by applying an arithmetic operator repetitively to the elements of the vector. The assignment  $\leftarrow$  operator has a special meaning in FSL. In addition to the usual function of separating the target variable from the body of the expression, it actually forces the FSL expression to be evaluated separately for every element in the target variable to avoid generalized arrays<sup>10</sup> as explained in the following example.

Example:

$$P.x \leftarrow + / Q.y \times R.z : S$$

where  $S$  is a predicate calculus expression such as:

$$P.v = Q.w \wedge R.z = c$$

This FSL expression implies that for every row  $i$  in  $P$ ,  $\{Q.y, R.z\}$  pairs satisfying the expression  $S$  are retrieved and fed into APL vectors named  $Q.y$  and  $R.z$ . Multiplication of these two vectors and summation of all the elements in the product vector produce the  $i$ th value in the vector  $P.x$ . The number of elements in vectors  $Q.y$  and  $R.z$  are different for every element of  $P.x$ ; hence it is possible to view the total structure as a generalized array where vectors along a particular dimension have varying number of elements (Figure 1). Another difference from standard APL is the ability to treat character strings as single elements rather than vectors.

The following examples illustrate the flexibility and convenience of FSL; given the information base of a manufac-

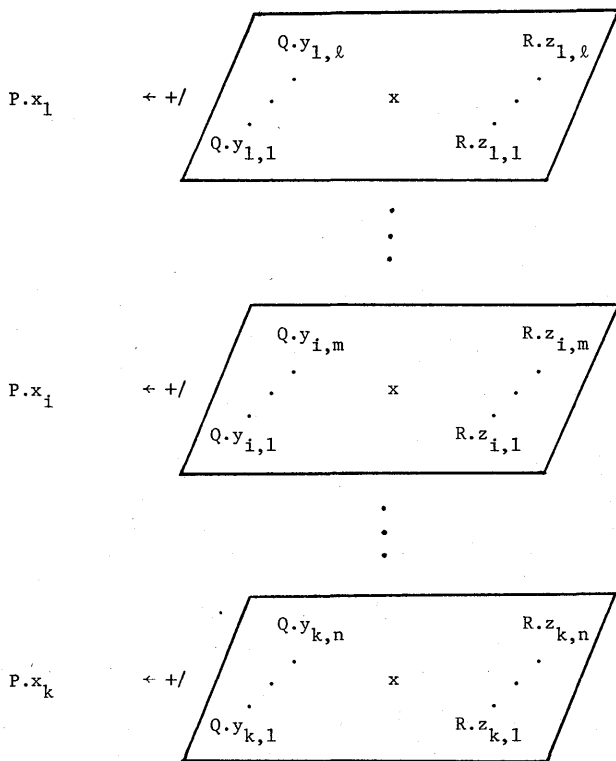


Figure 1—An FSL expression visualized as a generalized array and interpreted as a set of simple arrays.

turing environment where a number of projects receive supplies of parts from a number of suppliers.

SUPPLIER (s#, name, location, dr, cost) contains supplier number, name, location, discount rate, and the total cost of all supplies by the supplier.

PROJECT (j#, location, tec, cost) contains project number, location, total estimated cost and cost to date.

PART (p#, type, price, weight) contains part number, type, price and weight.

SUPPLY (l#, s#, p#, j#, quantity, cost, date) contains supply number, supplier number, part number supplied, project number receiving the supply, quantity, cost, and date received.

ROUTE (origin, destination, distance, cost) contains the distance between any two locations denoted by origin and destination and unit transportation cost per mile per pound on each route.

All cost fields contain derived data and all others are observed.

The following queries and functions are first expressed in English, then in FSL:

- a. Compute the number of suppliers in Milwaukee.  
 $\rho$  SUPPLIER. s#: SUPPLIER.location = 'MILWAUKEE'
- b. Compute the average weight of all parts of type A.

- + / PART.weight  $\div$   $\rho$  PART.weight  $\div$   $\rho$  PART.type = 'A'
- c. SUPPLY.tc is the transportation cost for each supply and depends on the distance, unit transportation cost and the total weight transported.  
 $SUPPLY.tc \leftarrow ROUTE.distance \times ROUTE.cost \times SUPPLY.quantity \times PART.weight: \exists PROJECT \exists SUPPLIER SUPPLY.s\# = SUPPLIER.S\# \wedge SUPPLIER.location = ROUTE.origin \wedge SUPPLY.j\# = PROJECT.j\# \wedge PROJECT.location = ROUTE.destination \wedge SUPPLY.p\# = PART.p\#$
- d. SUPPLY.dr is discount rate applying to each supply and it is equal to the discount rate of the supplier involved, or 5 percent more if the quantity is greater than 100.  
 $SUPPLY.dr \leftarrow (SUPPLIER.dr: SUPPLY.s\# = SUPPLIER.s\#) + 0.05: SUPPLY.quantity > 100$
- e. SUPPLY.cost is the cost of each supply and depends on the quantity, price, discount rate and the transportation cost.  
 $SUPPLY.cost \leftarrow (SUPPLY.quantity \times PART.price \times (1 - SUPPLY.dr)) + SUPPLY.tc: SUPPLY.p\# = PART.p\#$   
 Obviously, this function is utilizing the two previous functions SUPPLY.tc and SUPPLY.dr, assuming that they are available in the function store of the information base. Otherwise the substitution would have to be done by the user.
- f. PROJECT.cost is the cost to date of each project and it is defined as the sum of all supply costs for that project.  
 $PROJECT.cost \leftarrow + / SUPPLY.cost: SUPPLY.j\# = PROJECT.j\#$
- g. Create a relation SUP(p#, j#, num) which contains the number of supplies of each part for every project.  
 $SUP \leftarrow PART.p\#, PROJECT.j\#: \exists SUPPLY PART.p\# = SUPPLY.p\# \wedge SUPPLY.j\# = PROJECT.j\# . SUP.num \leftarrow \rho SUPPLY.l\#: SUPPLY.p\# = SUP.p\# \wedge SUPPLY.j\# = SUP.j\#$
- h. Which locations are supplying which locations?  
 $SUPPLIER.location, PROJECT.location: \exists SUPPLY SUPPLIER.s\# = SUPPLY.s\# \wedge SUPPLY.j\# = PROJECT.j\#$
- i. Which locations are supplying which locations exclusively? (supplier supplying only one location)  
 $SUPPLIER.loc, PROJECT.loc: \forall SUPPLY SUPPLIER.s\# = SUPPLY.s\# \rightarrow SUPPLY.j\# = PROJECT.j\#$
- j. PROJECT.distance is the average distance of suppliers supplying each project.  
 $PROJECT.distance \leftarrow + / ROUTE.distance \div \rho ROUTE.distance: \exists SUPPLY \exists SUPPLIER PROJECT.location = ROUTE.destination \wedge PROJECT.j\# = SUPPLY.j\# \wedge SUPPLY.s\# = SUPPLIER.s\# \wedge SUPPLIER.location = ROUTE.origin$
- k. PROJECT.farthest is the farthest location supplying each project.  
 $PROJECT.farthest \leftarrow ROUTE.origin$



```

[ROUTE.distance i Γ/ROUTE.distance]: ∃SUPPLY
∃SUPPLIER PROJECT.location =
ROUTE.destination ∧ PROJECT.j# = SUPPLY.j#
∧ SUPPLY.s# = SUPPLIER.s# ∧
SUPPLIER.location = ROUTE.origin
or:
PROJECT.farthest ← ROUTE.origin: ∃SUPPLY
∃SUPPLIER ∀ROUTE' PROJECT.j# =
SUPPLY.j# ∧ SUPPLY.s# = SUPPLIER.s# ∧
SUPPLIER.location = ROUTE.origin ∧
PROJECT.location = ROUTE.destination ∧
(PROJECT.location = ROUTE'.destination ∧
SUPPLIER.loc = ROUTE'.origin →
ROUTE.distance ≥ ROUTE'.distance) .

```

### USER FRIENDLINESS

The convenience to the final user of an information system is a major concern in language design. Some previous research<sup>5,6</sup> reported behavioral work to compare different syntaxes in terms of user preferences. Syntax of a language undoubtedly plays a role in determining the level of convenience to and acceptance by the final user; however one has to keep in mind that different types of users with different skills and needs may have different preferences of syntax. A syntax geared toward the naive user may very well turn out to be very cumbersome and inconvenient for more sophisticated users as probably is the case in QUERY by EXAMPLE.<sup>6</sup>

A technique which invariably decreases the workload on the final user is abstraction. Abstraction involves decreasing the information content of queries, hence decreasing the effort required to form a query. This process can be called

'incomplete querying' since the queries acceptable by the system do not necessarily have all the information necessary to respond to the query. Incomplete querying obviously requires the system to provide the missing information to complete a query and it is accomplished by storing query segments—which may be queries themselves—and assigning them abstract names. A user, then, is free to use these abstract names to refer to query segments and build a complex query from these segments. Iteration in this process is permissible and actually is the real source of power.

Information Base draws heavily on the concept of abstraction by storing functional relationships as functions, and referring to these functions by name in forming more complex functions. A simple example of this was provided in section 2 where the previously defined variable SUPPLY.cost was used in a query and the definition of SUPPLY.cost was retrieved by the system from the information base. It may be claimed that the user has to provide all function definitions at some point in time; however even if the same user has to provide all function definitions, division of the task into independent segments simplifies the work to a great extent. An information Base Administrator may be employed to maintain the structure and given the task of defining the functions to further improve the situation.

Following the same philosophy, we will name predicate calculus expression segments as functions and keep them in function storage to further simplify queries. Natural candidates for this process are segments involving natural joins of two or more relations. Intelligent choice of names results in expressions close to English expressions in terms of ease of interpretation and construction by humans. The names used here start and end with relation names and imply an access path between those two relations:

```

SUPPLY_CONTAINING_PART }
PART_CONTAINED_IN_SUPPLY } ↔SUPPLY.p# = PART.p#

SUPPLY_RECEIVED_BY_PROJECT }
PROJECT_RECEIVING_SUPPLY } ↔SUPPLY.j# = PROJECT.j#

SUPPLY_BY_SUPPLIER }
SUPPLIER_OF_SUPPLY } ↔SUPPLY.s# = SUPPLIER.s#

SUPPLIER_SUPPLYING_PROJECT }
PROJECT_SUPPLIED_BY_SUPPLIER } ↔∃SUPPLY
SUPPLY_BY_SUPPLIER ∧
SUPPLY_RECEIVED_BY_PROJECT

SUPPLIER_ON_ROUTE ↔ SUPPLIER.loc = ROUTE.origin
PROJECT_ON_ROUTE ↔ PROJECT.loc = ROUTE.destination

SUPPLY_USING_ROUTE } ↔∃PROJECT ∃SUPPLIER
SUPPLY_RECEIVED_BY_PROJECT ∧
PROJECT_ON_ROUTE ∧
SUPPLY_BY_SUPPLIER ∧
SUPPLIER_ON_ROUTE

PROJECT_USING_ROUTE }
ROUTE_USED_BY_PROJECT } ↔∃SUPPLY ∃SUPPLIER
PROJECT_RECEIVING_SUPPLY ∧
SUPPLY_BY_SUPPLIER ∧
SUPPLIER_ON_ROUTE ∧
PROJECT_ON_ROUTE

```

Given the above function base, some of the examples of section 3 are simplified as follows:

- a.  $SUPPLY.tc \leftarrow ROUTE.distance \times ROUTE.cost \times SUPPLY.quantity \times PART.WEIGHT:$   
 $SUPPLY\_USING\_ROUTE \wedge$   
 $SUPPLY\_CONTAINING\_PART$
- b.  $SUPPLY.dr \leftarrow (SUPPLIER.dr:$   
 $SUPPLY.BY\_SUPPLIER) + 0.05: SUPPLY.quantity$   
 $> 100$
- c.  $SUPPLY.cost \leftarrow (SUPPLY.quantity \times PART.price$   
 $\times (1-SUPPLY.dr)) + SUPPLY.tc:$   
 $SUPPLY\_CONTAINING\_PART$
- d.  $PROJECT.distance \leftarrow +/ROUTE.distance \div \rho$   
 $ROUTE.distance: PROJECT\_USING\_ROUTE$
- e.  $PROJECT.farthest \leftarrow ROUTE.origin$   
 $[ROUTE.distance \div \Gamma/ROUTE.distance]:$   
 $PROJECT\_USING\_ROUTE$

CONCLUSIONS

A non-procedural design language is extremely useful in providing the application system designer with means to communicate the system requirements to other designers or to the machine without efficiency considerations. Procedure has to be introduced to the information system by someone—or preferably some system—who has the full knowledge of physical data structure, operating system, and the hardware configuration, and it has to be introduced with efficiency as the only concern. If there is interdependence between requirements and the efficiency attainable to meet those requirements, an iterative approach should be taken rather than combining the two tasks, simply because of the size, effort, and the number of different skills involved in systems development process. The interpretation of FSL expressions and automated implementation of systems require automated access path selection and automated generation of application algorithms. Optimization of this process is beyond the scope of this paper. The current implementation work is geared toward the generation of feasible implementations directly from the information base design. The implementation language is naturally APL.

BIBLIOGRAPHY

1. Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus," *Proc. of 1971 ACM-SIGFIDET Workshop on Data Description Access and Control*, ACM, New York, 1971.
2. Bracchi, G., Fedeli, A., and Paolini, P., "A Language for a Relational Data Base Management System," *Proc. Sixth Annual Princeton Conf. Inf. Science and Systems*, Mar. 1972, 84-92.

3. Fehder, P. L., "The Representation Independent Language," Res. Rep. RJ 1121, IBM Research Laboratory, San Jose, Calif., Nov. 1972.
4. Held, G. D., Stonebraker, M. R., and Wong, E., "INGRES. A Relational Data Base System," *Proc. 1975 AFIPS Nat. Computer Conf.*, 409-416.
5. Boyce, R. F., Chamberlain, D. D., King, F. W., III, and Hammer, M. M., "Specifying queries as Relational Expressions: The SQUARE Data Sublanguage," *Comm. ACM*, 18, 11, Nov. 1975, 621-628.
6. Zloof, M. M. and DeJong, S. P., "The System for Business Automation (SBA): Programming Language," *Comm. ACM*, 6, June, 1977, 385-396.
7. Iverson, K. E., "A Programming Language," John Wiley, New York, 1962.
8. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Comm. ACM*, 13, 6, June 1970, 377-387.
9. Codd, E. F., "Relational Completeness of Data Base Sublanguages," *Courant Computer Science Symposia*, Vol. 6, "Data Base Systems," edited by R. Rustin, Prentice Hall Inc., Englewood Cliffs, N.J., 1972.
10. More, T., Jr., "Axioms and Theorems for a Theory of Arrays," *IBM Journal of Research and Development*, 17, 135, 1973.
11. Ormancioglu, L., "Process Management System: An Automated Design Methodology for Management Reporting Systems," Ph.D. Dissertation, Northwestern University, 1979.

APPENDIX

APL operators in FSL

Operator	Monadic	Dyadic
+	Plus	Add
-	Minus	Subtract
×	Signum	Multiply
÷	Reciprocal	Divide
⌈	Ceiling	Maximum
⌊	Floor	Minimum
*	Exponential	Power
⊙	Natural Logarithm	Logarithm
	Magnitude	Residue
!	Factorial	Combinations
?	Roll	Deal
ρ	Size	Reshape
,	Ravel	Catanate
i	Index generation	Index of
φ	Transpose	Dyadic transpose
ε		Membership
↑		Take
↓		Drop
⊖	Reverse	Rotate
⊖	Reverse	Rotate first
'	Literal value	
/	Reduction	Compression
[ ]	Indexing	

For a detailed explanation of how to use these operators, reader is referred to any APL manual.<sup>7</sup>



# Comparing load & go and link/load compiler organizations

by WILLIAM L. WILDER

*Intermetrics Inc.*  
Washington, D.C.

## INTRODUCTION

Compilers usually produce either absolute code that is executed immediately upon conclusion of the compilation or object code that is transformed by a linking loader into absolute code. These compiler organizations will be called Load & Go and Link/Load. Both Load & Go and Link/Load compilers use a number of passes to translate the source program into absolute code. A pass reads some form of the source program, transforms it into another form, and normally outputs this form to an intermediate file which may be the input of a later pass.

Load & Go compilers can reduce the need for file communication between passes by using internal buffers to hold the intermediate representations of the source program during the compilation passes. Load & Go compilers can eliminate the need for linking by producing absolute code that remains in core, whereas Link/Load compilers can produce better absolute code than Load & Go compilers at the expense of increased compilation time. Both types of compilers are needed in different environments. Load & Go compilers can be used during program development where there are many more compilations than executions. Link/Load compilers can be used after program development is complete to minimize execution cost.

Comparisons using three pairs of Load & Go and Link/Load compilers for the SIMPL,<sup>1</sup> PL/1, and Pascal languages show that the cost to compile a program is reduced by the use of Load & Go compiler organizations, but the cost of executing the program is increased. The cost of executing any program is dependent upon the amount of data supplied or the number of iterations through the data. This data dependency can make a program more expensive to compile and execute with a Load & Go compiler than with a Link/Load compiler.

## METHOD OF COMPARISON

Four programs have been used in comparing the Load & Go and Link/Load compilers: bubble sort, Ackermann's function, removal of blanks from text, and Newton's method of approximating the square root. Each of these programs has been translated into the SIMPL, PL/1, and Pascal languages and runs on existing compilers at the University of

Maryland Computer Center. The comparisons between compilers for each program are on the basis of compilation cost (including link cost for Link/Load compilers), execution cost, total cost, and absolute code size (including library routines and run-time stack). Each of these programs has been run approximately ten times while developing the method by which the cost figures are gathered and code size determined. No significant differences could be detected from one run to the next, but the graphs and tables presented in this paper represent the final run of each program.

The bubble sort has been a standard algorithm commonly found in comparisons between languages and compilers. FORTRAN and SIMPL have been compared using sort programs<sup>2</sup> before and their compilation, execution, and total costs are nearly equal. Other FORTRAN and SIMPL-programs<sup>3</sup> have been compared and are also nearly equal in terms of execution cost and absolute code size. The other programs have been chosen on the basis of the language features they exercise. Ackermann's function exercises recursion, blank removal utilizes string manipulation, and Newton's method uses real number arithmetic. Both Ackermann's function and Newton's method<sup>4,5</sup> have been used before in comparing ALGOL and PL/1 compilers to each other. The blank removal program<sup>6</sup> has been used in comparing the Load & Go SIMPL compiler to the Link/Load SIMPL compiler.

Before analyzing the comparisons, the compilers for each of the languages should be described. The Load & Go SIMPL compiler<sup>7</sup> is a two-pass compiler that communicates between passes in core and the Load & Go PL/1 compiler is a three-pass compiler similar to the PL/C compiler.<sup>8</sup> In the Load & Go SIMPL compiler (SIMPL-RLG) and PL/1 compiler (PLUM), the absolute code produced by the compiler remains in core and can be immediately executed. The Link/Load SIMPL compiler is a three-pass compiler and the Link/Load PL/1 compiler is a four-pass compiler. In the Link/Load SIMPL compiler (SIMPL-R) and PL/1 compiler (PL1), the object code is written by the compiler to a file which is the input to the linking loader. The linking loader outputs the absolute code to a file that can be executed. The run-time support systems for each of these compilers allows the run-time stack to be enlarged dynamically when necessary during execution. The initial size of the run-time stack is .5K words for SIMPL and 4K words for PL/1.

Both the Load & Go Pascal compiler and the Link/Load Pascal compiler are one-pass compilers. The Load & Go Pascal compiler (PASLGO) writes absolute code to three files which are the input to an executor program. The Link/Load Pascal compiler (PAS) outputs the object code to a file which is the input to the linking loader. The linking loader outputs the absolute code to a file that can be executed. At the start of execution, the run-time stack and heap are allocated 6K words of space and the stack grows upward toward the heap as the heap grows downward toward the stack. The run-time stack and heap cannot be enlarged dynamically during execution, but the run-time stack and heap can be initially allocated either 16K words or 32K words of space as an option.

**COST COMPARISONS**

Comparisons of the compilation cost, execution cost, total cost, and absolute code size for each of the programs on each of the compilers are displayed as graphs in Figures 1-4 and the actual data can be found in Table I of the Appendix. The compilation cost comparisons for the four programs

show that programs are from 50 percent to over 300 percent less expensive to compile using Load & Go SIMPL and PL/1 than Link/Load SIMPL and PL/1. Notice that it takes longer to link a PL/1 program than to compile it using the Link/Load PL/1 compiler. The last compilation cost comparison shows that Load & Go Pascal is more expensive than Link/Load Pascal, which is unusual because Load & Go compiler organizations normally reduce the compilation cost of programs.

The execution cost comparisons show that the absolute code generated by the Load & Go compilers are from 25 percent to 300 percent more expensive than the Link/Load compilers. The biggest difference in execution cost is between the SIMPL compilers, but Load & Go SIMPL is faster than any of the PL/1 compilers or the Load & Go Pascal compiler. This is probably not a fair comparison because both PL/1 and Pascal are block-structured languages, while SIMPL is not.

The absolute code generated by Load & Go compilers is from 25 percent to over 300 percent larger than the Link/Load compilers. The biggest difference is again between the SIMPL compilers. On the UNIVAC 1100/40 at the University of Maryland, the execution cost of any program is some factor of CPU time multiplied by some factor of the amount of core used, e.g.

$$\text{EXECUTION COST} = \text{CPU TIME} * (\text{amount of CORE} / 32\text{K of CORE}).$$

This formula reflects not only the execution cost of the absolute code, but also the compilation cost of the compilers and linker.

**DATA DEPENDENCY**

The practicality of Load & Go compiler organizations in compiling and executing a program is dependent upon the data supplied to that program. Given enough data, the compilation cost advantages of Load & Go compilers are more than offset by the execution costs of its code. The next set of comparisons shows this data dependency, comparing the total cost of the Load & Go compilers and Link/Load compilers for the original amount of data to sixteen times the original amount of data. The original amount of data consists of 50 numbers for the bubble sort, a pair of numbers for Ackermann's function, 20 lines of text for blank removal, and 10 pairs of numbers for Newton's method. This data has been chosen because it reasonably exercises each program.

The total cost comparisons are displayed as graphs in Figures 5-8 and the actual data can be found in Table II of the Appendix. These graphs show that the total cost of all compilers increases evenly from the original amount of data to sixteen times the original amount of data. The total cost of Load & Go SIMPL and PL/1 are less expensive than Link/Load SIMPL and PL/1 for up to eight times the original amount of data. From this point, Load & Go SIMPL's total cost increases until it is more expensive than Link/Load SIMPL at sixteen times the original amount of data. Load

Cost Comparisons Figures (\* = Load & Go, + = Link/Load)

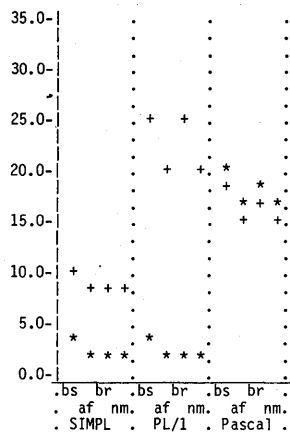


Figure 1—Compilation cost.

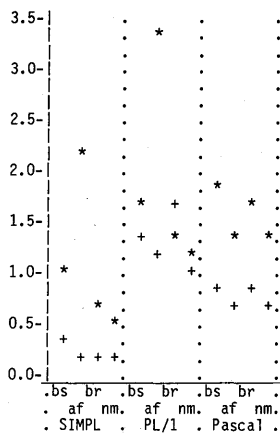


Figure 2—Execution cost.

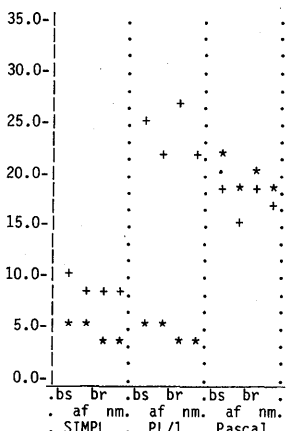


Figure 3—Total cost.

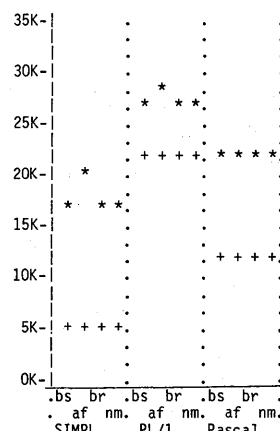


Figure 4—Absolute code size.

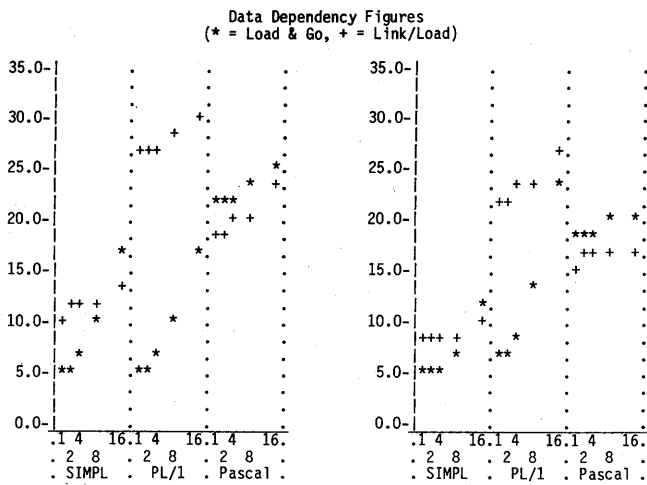


Figure 5—Bubble sort.

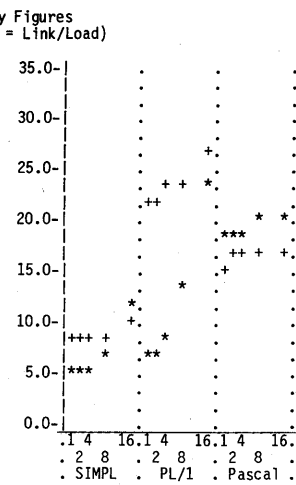


Figure 6—Ackermann's function.

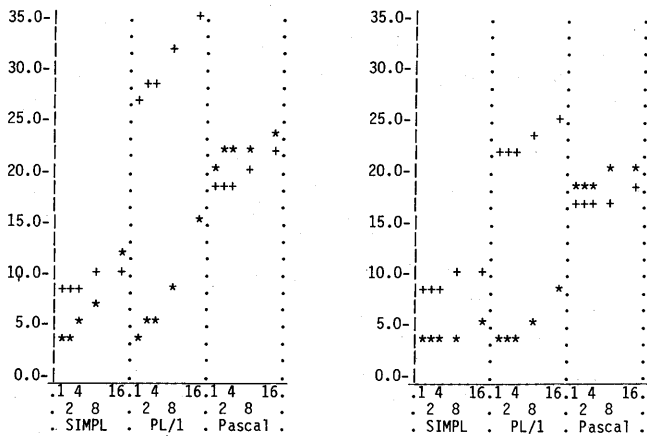


Figure 7—Blank removal.

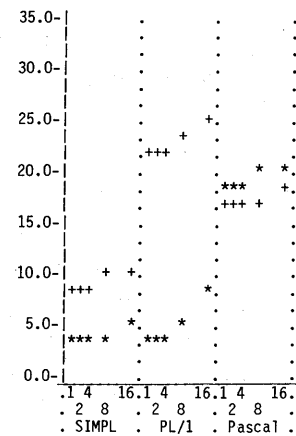


Figure 8—Newton's method.

& Go PL/1's total cost never becomes more expensive than Link/Load PL/1. Since Load & Go Pascal's compilation cost is never less expensive than Link/Load Pascal, Load & Go Pascal's total cost never could be less expensive than Link/Load Pascal.

CONCLUSIONS

These comparisons indicate that a Load & Go compiler should be about 50 percent less expensive than a Link/Load

compiler in compilation cost to make the use of Load & Go compiler organizations attractive. Allowing the intermediate representations of the source program to remain in core reduces the need for file references between passes, and thus the cost of compilation. Since linking is anywhere from 20 percent to 70 percent of compilation cost for a Link/Load compiler, a significant savings can be realized by a Load & Go compiler generating absolute code. Load & Go compilers would be most useful during program development where there are many more compilations than executions of each program. These program executions are typically on small representative sets of data.

A Link/Load compiler's execution cost should be at least 50 percent less expensive than a Load & Go compiler's to make the use of Link/Load compiler organizations competitive. The initial compilation cost for producing the better absolute code must be more than offset by execution cost savings. Since absolute code size affects the execution cost directly, the smallest possible set of library routines and runtime stack should remain in core during the execution of the absolute code. The ability to enlarge the run-time stack dynamically during execution helps achieve this goal. Link/Load compilers would be used when production runs start and there are many more executions than compilations of each program. These program executions are usually on larger and more numerous sets of data.

REFERENCES

1. Basili, V. R. and A. J. Turner, *SIMPL-T, A Structured Programming Language*, Palladin House Publishers, Geneva, Ill., 1976.
2. Basili, V. R., "Structured Programming Language for Compiler Writing: SIMPL-T," *Information Processing Society of Japan* 17, 3, 1976.
3. Basili, V. R., "A Transportable Extendible Compiler," *Software—Practice and Experience*, Vol. 5, 1975.
4. Wichmann, B. A., "Five ALGOL Compilers," *The Computer Journal*, Vol. 15, No. 1, 1971.
5. Wortman, D. B., P. J. Khaiat and D. M. Lasker, "Six PL/1 Compilers," Technical Report CSRG-36 Computer Systems Research Group, University of Toronto, November 1974.
6. Wilder, W. L., "Load and Go Techniques for Student Environments," Master's Thesis, Computer Science Department, University of Maryland, April 1978.
7. Wilder, W. L., "SIMPL-RLG: A Load-and-Go Compiler for the SIMPL Language," Computer Science Technical Report Series TR-658, University of Maryland, May 1978.
8. Conway, R. W. and T. R. Wilcox, "Design and Implementation of a Diagnostic Compiler for PL/1," *Communications of the ACM*, 16, 3, 1973.

## APPENDIX

TABLE I.—Cost Comparisons

	SIMPL		PL/1		Pascal	
	Load & Go	Link/Load	Load & Go	Link/Load	Load & Go	Link/Load
Bubble Sort						
Compile	3.4	7.0	3.2	11.6	20.0	14.3
Link	----	2.5	----	12.5	----	4.0
Execute	1.0	0.3	1.6	1.3	1.8	0.8
Total	4.4	9.8	4.8	25.4	21.8	19.1
Size	17.0K	4.5K	26.0K	21.0K	21.0K	12.5K
Ackermann						
Compile	2.5	5.9	2.3	7.5	17.1	11.3
Link	----	2.5	----	12.6	----	3.9
Execute	2.1	0.2	3.3	1.1	1.4	0.6
Total	4.6	8.4	5.6	21.2	18.5	15.8
Size	19.5K	4.5K	29.0K	21.0K	21.0K	12.5K
Blank Removal						
Compile	2.5	6.0	2.2	7.8	19.0	13.8
Link	----	2.3	----	17.8	----	4.1
Execute	0.6	0.2	1.4	1.7	1.7	0.8
Total	3.1	8.5	3.6	27.3	20.7	18.7
Size	17.0K	4.5K	26.0K	21.0K	21.0K	12.5K
Newton						
Compile	2.5	6.0	2.2	6.8	17.0	11.7
Link	----	2.6	----	13.4	----	4.0
Execute	0.5	0.2	1.1	1.0	1.3	0.7
Total	3.0	8.8	3.3	21.2	18.3	16.4
Size	17.0K	4.5K	26.0K	21.0K	21.0K	12.5K

TABLE II.—Data Dependency

	SIMPL		PL/1		Pascal	
	Load & Go	Link/Load	Load & Go	Link/Load	Load & Go	Link/Load
Bubble Sort						
Data	4.3	10.8	4.7	26.2	21.3	18.5
Data2	5.0	11.0	5.3	26.6	21.5	18.7
Data4	6.6	11.2	6.8	27.3	22.3	19.3
Data8	9.6	11.6	10.0	28.6	23.5	20.4
Data16	16.0	12.5	17.0	29.3	25.7	22.6
Ackermann						
Data	4.8	8.9	5.8	21.8	18.6	15.8
Data2	5.0	9.0	6.8	22.1	18.7	15.9
Data4	5.8	9.1	9.0	22.6	18.9	16.0
Data8	7.3	9.3	13.5	24.0	19.4	16.3
Data16	11.1	9.9	23.0	26.4	20.2	17.1
Blank Removal						
Data	3.5	8.9	3.7	27.4	20.7	18.0
Data2	3.9	9.0	4.4	28.0	21.0	18.2
Data4	4.9	9.1	5.7	29.0	21.4	18.6
Data8	6.9	9.5	8.4	31.2	22.3	19.4
Data16	11.0	10.8	14.5	36.2	24.3	21.0
Newton						
Data	3.1	9.0	3.3	21.8	18.5	16.1
Data2	3.2	9.1	3.6	22.0	18.6	16.2
Data4	3.4	9.2	4.0	22.3	18.8	16.4
Data8	4.0	9.3	5.2	22.9	19.3	16.8
Data16	5.7	9.8	8.1	25.1	20.5	18.0

# A link between polygon and grid representations of land resource information systems

by DEVON NICKERSON

*Logging Systems Group, U.S. Forest Service Pacific Northwest Region  
Portland, Oregon*

Land-use planners, foresters, agricultural scientists, and others involved in land-based resource decision-making have been presented in recent years with a computer tool for examining land attributes: the resource data base. Innumerable developments have been made in this field. A few examples are:

WRIS—the Wildland Resource Inventory System, developed and in use within the U.S. Forest Service California Region.

COMARC—a commercial geographic data base system, developed by Comarc Design Systems of San Francisco, California.

PLOT—the Polygon Layer Overlay Technique, developed by the U.S. Forest Service Pacific Northwest Region, and currently used by the U.S. Fish and Wildlife Service.

MAP—the Map Analysis Package, developed at the Yale School of Forestry and Environmental Studies.

TRI—the Total Resource Inventory system, developed and in use by the U.S. Forest Service Pacific Northwest Region.

Although there is a large array of computerized resource data bases in existence, the objectives of all are nearly the same. First, resource data bases seek to capture relevant resource attributes in "layers." Resource attribute layers may include soil type boundaries, ownership boundaries, timber types, wildlife range limits, elevation zones, slope zones, aspect zones, political boundaries and many more characteristics of the land. Second, resource data bases seek to store this information, recall it, update it, and display it in comprehensible form. Third, resource data bases allow the retrieval and display of land possessing a set of common attributes, from which the resource manager can make some conclusions. For example, a forester may query a resource data base for a composite display of all land under National Forest administration, in Sierra County, of old-growth mixed conifer timber type, between 4000 and 5000 feet elevation, and on south-facing slopes; this set of attributes may represent good timber-harvest opportunities, but difficult reforestation problems. It can be seen that one computational

task of resource data bases is to calculate the unions or intersections of a host of information layers.

Resource data bases suffer a sharp division on one major point: the form in which terrain data is stored and manipulated. There are two basic structures. *Polygon* data bases handle land resource attribute compartments as a set of (*X*, *Y*) coordinates, representing points along the compartment boundary (Figure 1). *Grid* data bases divide the land into regular grid cells, and handle an attribute compartment by tallying the cells included within the compartment (Figure 2).

Significant differences in computational tractability occur between Polygon and Grid data base systems. While most resource data-collection methods correspond to the Polygon format—for example, ground survey techniques and map- or photo-digitizing—it is a difficult undertaking to devise an algorithm to compute the intersection of two or more irregular polygons. The Grid approach, on the other hand, lends itself to computational ease: two or more layers can be intersected with simple logical operators on a cell-by-cell basis. First, however, each layer grid must be compiled, usually derived from resource data collected in polygon form. In addition, the resolution of grid resource data is entirely dependent on the size of the individual grid cell. If each grid cell represents a hectare of land, then attributes or fragments of attributes of smaller size or irregular outline either fail to show up or are generalized into one-hectare square shapes. Also, data storage requirements may be much greater for Grid systems than for Polygon systems.

A method for efficiently moving from Polygon resource attribute depiction to Grid attribute depiction would be useful indeed. By doing so, the resource manager would be free to use any of the various resource data base systems, and could take advantage of the Polygon systems' high resolution and storage efficiency, and the Grid systems' computational ease and efficiency. Such a link procedure should operate at any scale, should be simple to effect in program software, should be computationally easy and efficient, and should contribute to the graphical representation of the resource attribute layer.

While developing a software package for simulating the visual effects of timber harvest activities, a hardware-de-



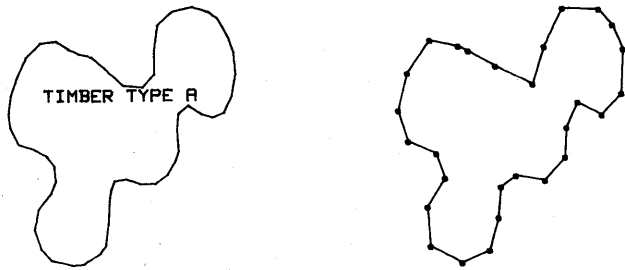


Figure 1—Resource attribute compartments are recorded as boundary point coordinates in Polygon-type data bases.

pendent link technique between Polygon resource bases and Grid resource bases became apparent. The Visual Simulation program package uses a graphics desktop computer system with a raster-scan Cathode Ray Tube (CRT) as an output device. Alphanumeric or graphics images may be displayed on the CRT. The image is composed of discrete phosphor dots, or picture elements. An internal buffer records an exact numeric model of the graphics raster, by retaining an “on” bit for every illuminated picture element. The graphics raster has  $455 \times 560$  dots, or 254,800 bits of information. This represents nearly 32,000 bytes of “bonus” information storage, since the graphics buffer is separate from user-accessible memory. It was a desire to exploit this extra storage space that led to discovery of the Polygon-Grid link technique.

Raster-scan graphic images tend to be a little lacking in resolution, since diagonal lines must be approximated by illuminating the discrete dots lying more-or-less along the line. Even so, the dots are very tiny, and there are more than a quarter million dots on the CRT. If an image of a resource attribute compartment could be portrayed on the graphics CRT, each included picture element could be thought of as a grid cell possessing that attribute. Moreover, by plotting the polygon attribute boundary to any desired scale, and coloring-in within the attribute boundary, the resolution of the grid is controllable. The graphics buffer then represents a storable, retrievable numerical model of the attribute in grid form. Since all or part of the graphics buffer can be transferred to the user-accessible memory, binary logical operators can be used to rapidly calculate unions or intersections of various attributes depicted in this fashion.

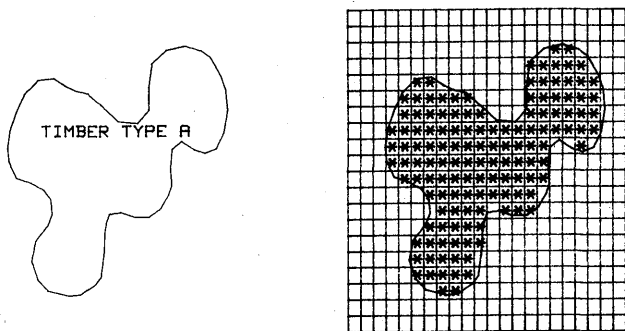


Figure 2—Resource attribute compartments are recorded as included grid cells in Grid-type data bases.

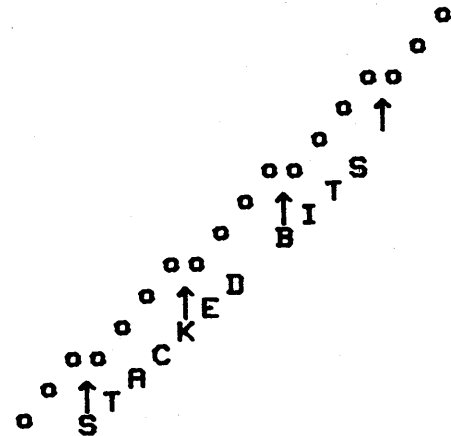


Figure 3—Line segments inclined at less than 45 degrees result in bits stacked in the horizontal direction.

Upon retrieval or combination of attributes, the result is instantly displayable in graphic form.

As with many a good idea, there is a catch. Almost any six-year-old equipped with a crayon can color-in inside the lines. With a computer this is not so easy. One must start with a boundary and turn on all the picture elements inside. The logical approach would be to move sequentially down the graphics raster, examining each horizontal line in turn. Moving from left to right along the line, one would realize he was “inside” upon encountering the first “on” bit, which would represent first crossing the boundary *into* the attribute compartment. From that point on, every bit would be turned “on,” until the next “on” bit was encountered, which would represent crossing the boundary *out of* the attribute compartment. In this manner, very complicated shapes with reflex curves, fingers, and holes could be colored-in. The catch is that bits become “stacked” in the horizontal direction when a boundary segment is drawn at an inclination of less

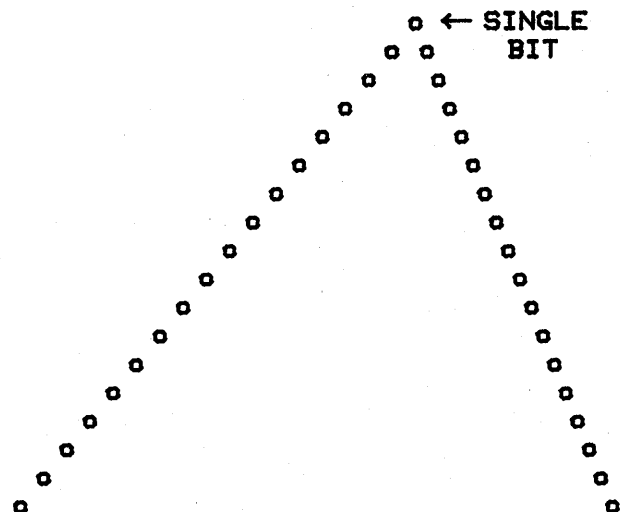


Figure 4—A vertical change-of-direction leaves a single bit.

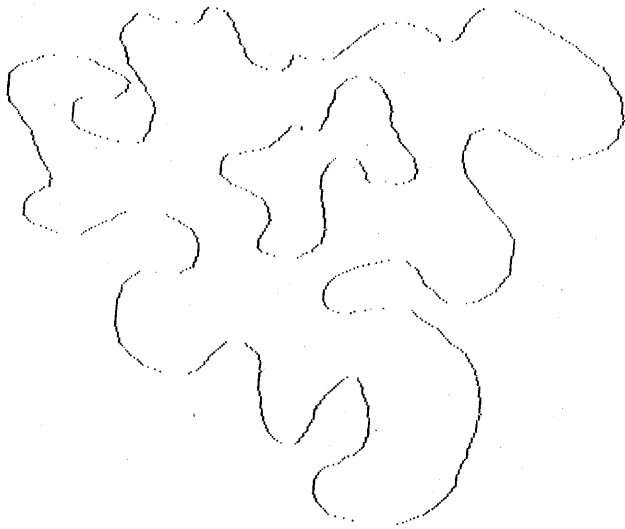


Figure 5—An attribute boundary polygon, plotted without stacked or single bits.

than 45 degrees from horizontal (Figure 3). The color-in technique described above would think “entering” at the first bit, “leaving” at the second bit, and would fail to color-in the horizontal line beyond the second bit. In addition, at points where the boundary changes direction vertically, only a single bit may be “on” (Figure 4). The color-in technique would commence coloring-in upon encountering this bit, and would leave a spurious trail of “on” bits extending to the right.

The solution is to draw the boundary such that neither anomalous event occurs. One such boundary is shown in Figure 5. Bit-stacking is eliminated by controlled plotting of

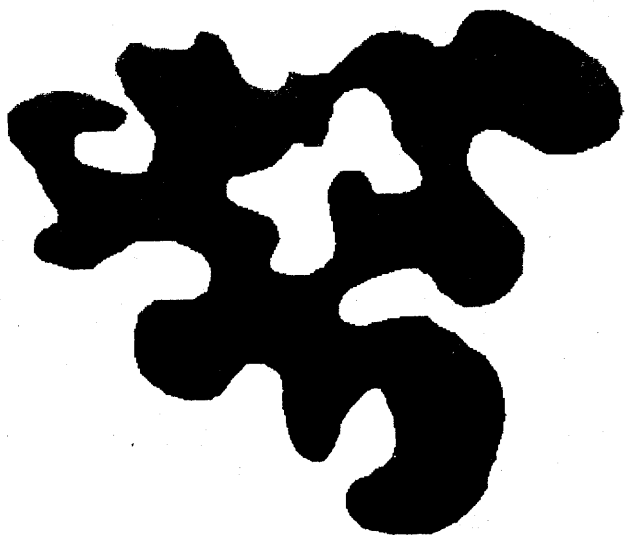


Figure 6—The attribute boundary polygon is colored-in, yielding an attribute grid composed of individual picture elements.

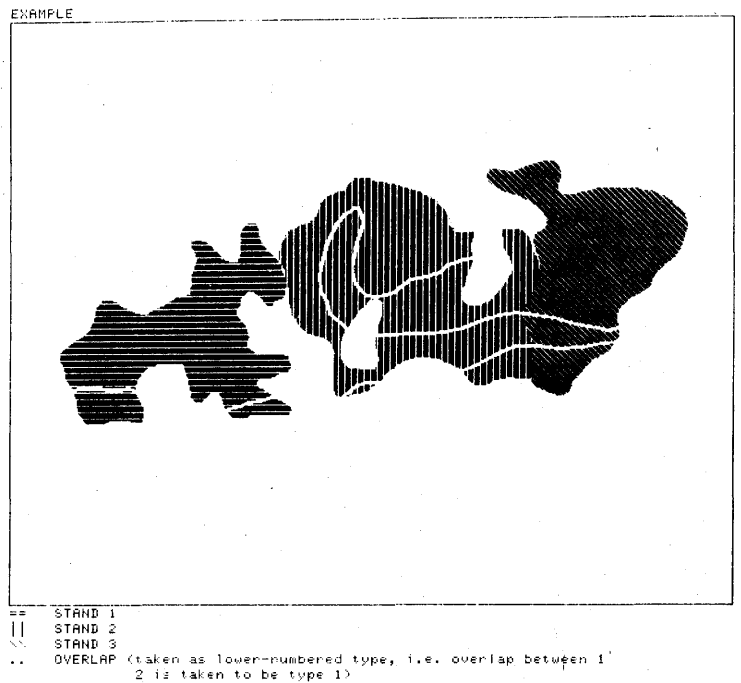


Figure 7—A composite of attribute grids and crosshatch graphics.

line segments inclined less than 45 degrees. Single bits at vertical changes-of-direction are deleted. The color-in procedure can be performed exactly as described above, with results as seen in Figure 6.

The Visual Simulation routine used this technique to model partial-cut timber harvest settings. Boundaries of distinct timber types are digitized, producing an attribute polygon. The color-in routine creates an attribute grid. Each grid cell or picture element represents a tree crown. By inserting a random-number selection mechanism, various intensities of partial timber removal can be modelled. Several distinct timber type layers were combined using an EXCLUSIVE OR operator. A layer of unforested openings (clearcuts, meadows, water bodies, and road right-of-way) was added, using the BINARY COMPLEMENT and the BINARY AND operators. Crosshatching graphics, to identify the separate timber types, were added using the binary AND operator. The composite graphic, generated in a single pass across seven grid layers, is shown in Figure 7. The plan-view from this composite graphic was used, via the graphics buffer, to produce a simulation of the timber stand as seen in true perspective (Figure 8).

The technique described in this paper is an attempt to turn a computer hardware peculiarity to useful advantage. The graphics buffer of the desktop computer system described can certainly be mimicked within any large computer mainframe. More exciting would be a graphics desktop system used to prepare resource layer information as described, then passing the layer in both Polygon and Grid form to a major mainframe for storage or manipulation. Output would return to the graphics system to be formatted for display.

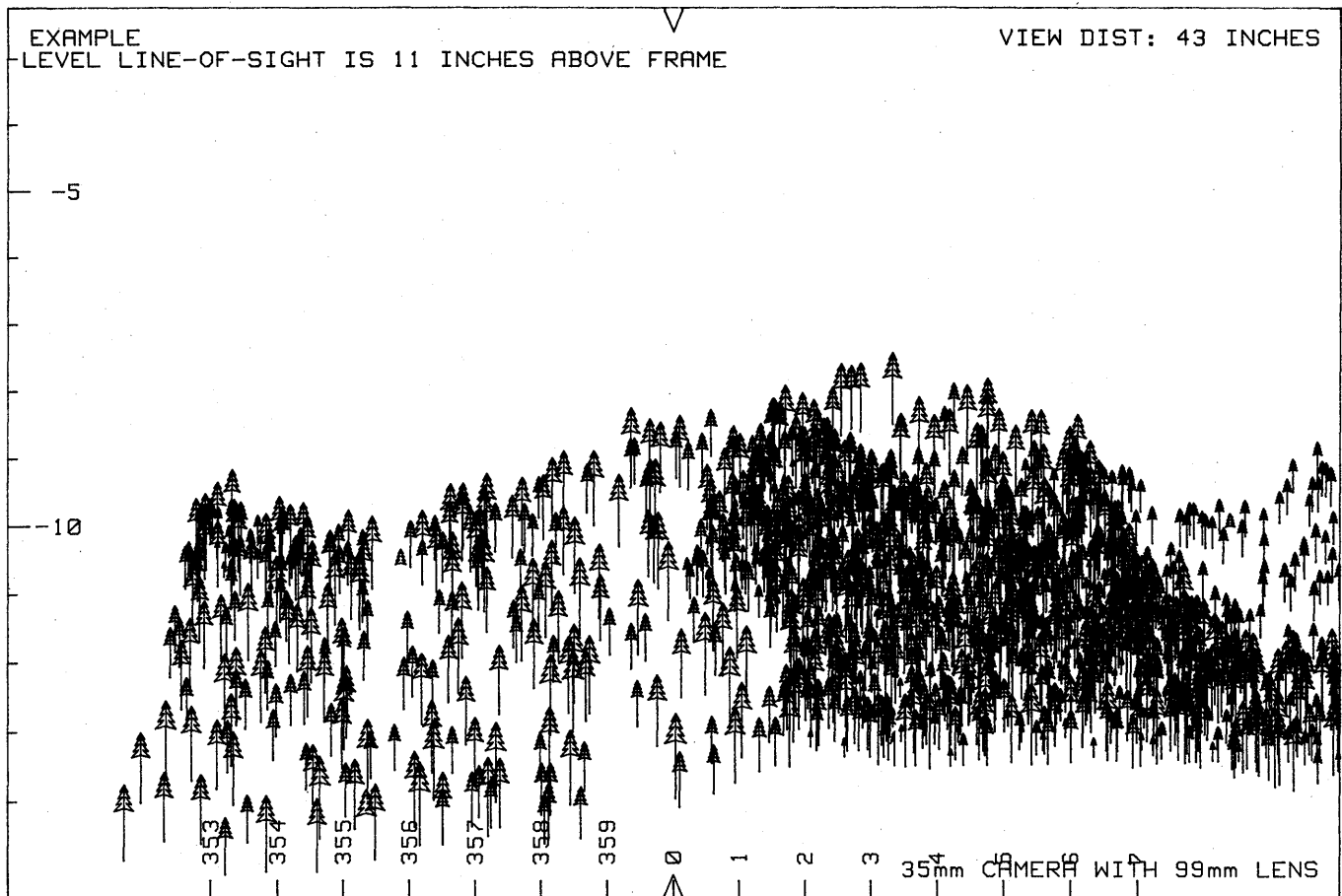


Figure 8—Partial-removal timber harvest simulation, derived from the composite attribute grid in Figure 7. Note many-layered unharvested stand, shelterwood stand (left, top center), and thinned pole-size timber (right).

It is for the sake of these future possibilities that the Polygon-Grid link technique is described in this paper.

#### REFERENCES

Salmen, L., et al., *Comparison of Selected Operational Capabilities of Fifty-Four Geographic Information Systems*, Western Governors Policy Office, Fort Collins, Colorado, 1977.

Schwarzbart, G., et al., *Analysis of Computer Support Systems for Multi-functional Planning, Report III*, U.S. Forest Service Management Sciences Staff, Pacific Southwest Region, Berkeley, California, 1976.

Tomlin, C. D., *Cartographic Modeling Techniques in Natural Resources Management*, doctoral dissertation, Yale School of Forestry and Environmental Studies, New Haven, Connecticut, 1979.

Tomlin, C. D. and Berry, J. K., *Map Analysis Package*, Yale School of Forestry and Environmental Studies, New Haven, Connecticut, 1979.

Young, R. W., *Evaluation of a Commercial Geographic Data Base for Storage and Retrieval of Forest Insect and Disease Information*, U.S. Forest Service Methods Application Group, Davis, California, 1979.

# Risk analysis in the 1980's

by JEROME LOBEL

Honeywell Information Systems, Inc.  
Phoenix, Arizona

## INTRODUCTION

The application of scientific procedures to the study and evaluation of information and communications systems risks is still in its infancy. Hopefully, before the end of this decade we will see major breakthroughs both in improved techniques and greater utilization of Risk Analysis procedures by computer users. On the other hand Risk Analysis (also sometimes called Threat or Vulnerability Analysis) has real merit even by today's standards. The problem is that many organizations have still to be convinced as to its potential.

## THE PAST—RISK ANALYSIS LIMITATIONS

Risk Analysis attained a certain degree of popularity as a result of a report written for the Federal Information Processing Standards Task Group 15, Computer Systems Security, of the United States Department of Commerce National Bureau of Standards in 1975.

Although recognized as a potentially valuable evaluation tool authorities generally did not present it as a panacea for relieving the ills of an electronic data processing system. Typical systems problems such as fraud, theft, embezzlement, malicious damage, unauthorized access, natural disaster, accident, or an operations interruption or stoppage were considered to be too complex to be resolved by relatively simple mathematical or statistical solutions.

Other criticisms of Risk Analysis methodology included:

- The owners and users of information systems (the people from whom survey data is usually obtained) often do not have enough detailed knowledge as to how their systems work or where their systems work or where their vulnerabilities are located to provide adequate or sufficiently accurate information.
- It is difficult if not impossible (or impractical) to survey 100 percent of an exhaustive list of system vulnerabilities.
- Estimates of event occurrence (the probability of an event occurring) or its cost may be too imprecise to be reliable.
- Some information system threats are not quantifiable in monetary terms (i.e. national security information compromises, loss of public services, etc.).

- The most fallible part of most information systems, the human factor, is too unpredictable and uncontrollable to measure.

In spite of these limitations, many organizations began to use Risk Analysis or some variation of it to get a better "handle" on their information system vulnerabilities.

## THE PRESENT—STANDARD PROCEDURES

In practice, there are many variations in Risk Analysis technique and approaches. The basic process of Risk Analysis however tends to follow the following four steps:

1. A survey is made of an organization's risks associated with its most essential assets, typically its people, information and facilities. Normally the data gathered during the survey or study includes:
  - The identification of potentially injurious or disastrous events,
  - Estimates of the frequency of occurrence associated with risk events (Figure 1).
  - Estimates of cost (usually in money) of the loss per incident of event occurrence (Figure 2).Special statistical tables are often used to permit even gross estimates of time or cost to be mathematically useful.
2. Calculations are made based upon the input data (estimates made during the survey) and result in the derivation of an expected *annual loss* from the occurrence of a particular event (Figure 3).
3. A detailed evaluation of each event or problem area is made to identify the best known preventative measures and their associated costs.
4. A return-on-investment (ROI), pay back calculation or other comparative measurement technique is used to evaluate the reasonableness of spending time, money or energy to reduce a particular risk. Risk Analysis studies usually result in some form of management decision. As an example, if a particular risk prevention measure is deemed too expensive or not practical, a decision may be made to "tolerate" the risk.

Advocates claim that the final result of a formal Risk

E/Y = FREQUENCY OF EVENT	
P = PROBABILITY	
<b>EXAMPLE:</b>	
LET P = 0	IF PRACTICALLY NEVER
= 1	IF ONCE IN 300 YEARS
= 2	IF " " 30 YEARS
= 3	IF " " 3 YEARS
= 4	IF " " 100 DAYS
= 5	IF " " 10 DAYS
= 6	IF " " 1/DAY
= 7	IF 10/DAY
= 8	IF 100/DAY
(IF 3 YEARS = 1000 DAYS)	

Figure 1—Probability of frequency estimation table.

Analysis survey will be a set of informed management decisions, possibly several magnitudes better than the intuitive guess-work that might have otherwise taken place. It is also proposed that Risk Analysis should be a dynamic or on-going process which is repeated or periodically updated. Its advocates also claim that it is one of the few systematic approaches to resolving potentially dangerous problems associated with certain data processing and communications systems.

Risk Analysis studies may be performed by special data processing project teams, internal audit staffs, professional security staffs or outside consultants, just to name a few of the organizations often called upon to do the job.

### THE FUTURE—TAILORING PROCEDURES TO SATISFY REAL WORLD NEEDS

The idea that Risk Analysis, as a way of measuring and correcting information system threats, might be overlooked by computer users led to a survey of 250 organizations that had already been exposed to the methodology. The objective of the survey was to analyze the extent to which formal Risk Analysis procedures were being used by these organizations and the nature of the benefits that were being derived. Fifty-eight responses were received and were tabulated in the results.

Altogether, there were ten questions in the survey. A number of the questions had multiple parts and permitted the respondee to comment on significant issues. All responses were based upon organizational as opposed to individual experiences.

<b>EXAMPLE:</b>			
LET COST = 12	IF LOSS IS	\$ 3,333,333	
= 11	" " "	\$ 1,000,000	
= 10	" " "	\$ 333,333	
= 9	" " "	\$ 100,000	
= 8	" " "	\$ 33,333	
= 7	" " "	\$ 10,000	
= 6	" " "	\$ 3,333	
= 5	" " "	\$ 1,000	
= 4	" " "	\$ 333	
= 3	" " "	\$ 100	
= 2	" " "	\$ 33	
= 1	" " "	\$ 10	
= 0	" " "	\$ 3	
			<b>L/E = LOSS PER EVENT</b>

Figure 2—Estimate of order of magnitude table.

The following is a list of the questions, the responses tabulated, and this author's comments and conclusions regarding the response to each question:

1. Has your organization implemented a formal Information Systems Risk Analysis program at any time?  
 Response: Yes = 10 No = 48  
 Comment: The low response of 21 percent (organizations with formal Risk Analysis Programs) indicates that, at the very least, Risk Analysis has not yet met with wide acceptance as a means of identifying and correcting information system threats.
2. Has your organization used the *formal* Risk Analysis technique for studying information system weaknesses at any time in the past?  
 Response: Yes = 12 No = 46  
 Comment: For those organizations that have im-

<b>EXAMPLE</b>	
L/E =	EXPECTED LOSS PER EVENT
E/Y =	EXPECTED FREQUENCY OF EVENT
L/Y =	EXPECTED AVERAGE LOSS PER YEAR
$L/Y = (L/E) (E/Y)$	

Figure 3—Expected average loss per year calculation.

plemented the program, it appears that they must have either attained some degree of success or that the program is only in its initial phase of implementation. (Only one organization indicated the program was a washout.)

3. If your organization has used Risk Analysis, would you say that the extent to which it was applied was:

Response:	Check one only
a. Extensive (all or most systems or applications)?	9
b. Moderate ( $\frac{1}{4}$ to $\frac{3}{4}$ )?	3
c. Occasional (less than $\frac{1}{4}$ )?	10

Comment: Although there appears to be some discrepancy between the answer to this question and the previous questions, it would seem that the majority of organizations that implemented a *formal* Risk Analysis program tended to go all the way—that is surveyed and evaluated all applications as opposed to only part of their information system. (It is likely that organizations that checked part c., "Occasional," probably did not consider their prior use of Risk Analysis type studies as being a "formal" application of the methodology.)

4. If your organization implemented a Risk Analysis program, how good were the results?

Response:	
a. Excellent (est. savings in excess of \$.5 million)	0
b. Good (est. savings between \$100,000 and \$.5 million)	2
c. Fair (est. savings less than \$100,000)	0
d. Poor (savings could not be identified)	9

Comment: The rather negative outcome indicated by the responses to this question can be indicative of generally poor results, poor follow-up, measurement difficulty, or it may have been too early for Risk Analysis users to measure their results. Also, there were a number of questionnaires sent back with comments to the effect that the reason for their organization doing Risk Analysis was not necessarily to obtain monetary cost savings. They said that their main objective was simply to identify risks and implement preventative measures.

5. If your organization has not used Risk Analysis, which of the following reasons probably accounted for this:

Response:	
a. Lack of management support	18
b. Lack of adequate information on the technique	17
c. Technique lacks rigid discipline	2
d. Other techniques easier to use	3
e. Could not determine a Risk Analysis Survey would accomplish anything	13

Comment: The reasons given for not implementing a Risk Analysis program indicate that potential users want a lot more proof that the effort and results will probably be worthwhile. So it seems likely that we will not see a significant increase in the use of formal Risk Analysis programs to reduce information system threats until more organizations report positive results, or possibly develop and use other techniques which get the job done better. There is also a strong indication that many potential users of Risk Analysis are looking for more educational information and articles on Risk Analysis methodology and its practical application.

6. What would you say is the strongest argument for doing a Risk Analysis study?

Response:	Check as Appropriate
a. Quantification of system risks and priorities	30
b. Focus attention on high risk areas	29
c. Confirmation of previous threat studies	2
d. Alerting of the organization	30
e. Management participation	11
f. No other technique available	0
g. The resulting action steps	9

Comment: It is interesting to note that responses to this question indicate a greater interest in the communications and quantification value of risk analysis compared to the final outcome or results of implementing study recommendations. This may mean that many people consider Risk Analysis more of an education and planning tool than the final answer as to where to apply resources to minimize or eliminate information system vulnerabilities.

7. If your organization is not using Risk Analysis techniques, are you using some form of substitute program or procedure?

Response: Yes = 11 No = 27

Comment: The number of yes answers are significant inasmuch as almost as many Risk Analysis-using organizations reported they were using some form of *modified* procedure for evaluating systems risks. (See the next question.)

8. If you have used or are presently using Risk Analysis, have you modified or improved on the standard procedure in order to get better results?

Response: Yes = 10 No = 16

Comment: Again, the large number of organizations that reported that they were using some modified form of Risk Analysis to study their system vulnerabilities seems to attest to the need for organizations to tailor whatever procedure they elect to use to their own needs and purposes.

9. If you have used or are using Risk Analysis, have you developed any new or unique survey forms or cal-

culational procedures that you could share with other interested organizations?

Response: Yes = 5 No = 23

Comment: Although only a few of the responding organizations felt that they were in a position to contribute to the state-of-the-art (at the time of this survey), the ideas that were sent in were extremely interesting. (See the next chapter—Risk Analysis Enhancements.) As an example, a number of organizations went to some form of unique procedure for prioritizing or weighting risks related to the needs of their own organization. This helped to partially reduce the amount of time and precision required to estimate risk relevancy and monetary cost savings. As a result, a Risk Analysis study using an alternate procedure might be more useful to a particular organization. Furthermore, modified approaches probably work better where the inherent nature of the system makes it difficult or impractical to utilize monetary values as a basic measurement criteria.

10. If your organization has not performed a Risk Analysis Survey or other similar study of your information system vulnerabilities, what are the possibilities of a program being implemented sometime in 1979?

Response: Excellent 10  
 Probable 19  
 Negative 14

Comment: The majority (29) of the responding organizations that had not yet already initiated some form of formal Risk Analysis program indicated that they would probably do so prior to the end of this year (1979). To some extent, this is surprising in the light of the answers given to the other questions in the survey. One conclusion that could be drawn in line with this response is that computer users recognize that systems abuses and risks do not appear to be diminishing, and therefore some type of action program will soon be needed. The problem may be which risk evaluation program should be implemented and when?

**RISK ANALYSIS ENHANCEMENTS**

Thanks to the generous cooperation of the organizations that responded to the Risk Analysis Survey, the following ideas are presented as examples of techniques that might be used to modify or enhance a Risk Analysis program.

*Example 1*

Objective: A simpler, less expensive procedure—This computer user reported that they operated a medium-sized installation, and didn't have the manpower to implement a "formal and extensive" Risk Analysis program. Their so-

lution was to develop a simplified data gathering form (Figure 4), which they felt short-cutted a more expensive and time-consuming study.

*Example 2*

Objective: Shorten the Risk Analysis data gathering cycle and expedite evaluation of more critical computer applications—This organization initially used the evaluation procedure published by the Institute of Internal Auditors in their Systems Auditability and Control-Audit Practices guide. They reported that they didn't have time, however, to compile all of the required data, but determined that they could get by with three indexes and an overall summary. (See Figure 5.) The indexes are referred to as the: (1) Major Systems Index, (2) Company Assets Index, and (3) Critical Systems Index.

*Example 3*

Objective: Modify standard Risk Analysis procedures to more clearly distinguish the severity of impact of different classes of hazards—This organization developed an eight point "degree of loss" index (See Figure 6), and a special form to permit a more quantitative review of information system hazards.

CRITICAL
----------

**COMPUTER SECURITY ANALYSIS  
(DATA GATHERING FORM)**

System name \_\_\_\_\_ System Identification \_\_\_\_\_

Description of system: \_\_\_\_\_

← MANUAL SYSTEMS
← COMPUTER SYSTEMS →

Input from							
Output to							

Effect of disruption of service: \_\_\_\_\_

Alternate method: \_\_\_\_\_

Effect of loss/destruction of files: \_\_\_\_\_

	Fire	Power	Earth	Sabotage	Fraud	Error
Probabilities of occurrence						
Recovery plan established						
Countermeasures taken						

Remarks and notes: \_\_\_\_\_

			<b>COST OF LOSS</b>	
<b>TYPE OF LOSS</b>			<b>FILES</b>	<b>BUSINESS</b>
PERMANENT				
TEMPORARY				

Figure 4.

**RISK IMPACT INDICES**  
(Critical Scale is 1 to 10 with 10 being the high value or most critical condition)

**Major Systems Index**

9-10	Over 60 programs or 100 man months of maintenance, or 10,000 computer hours annually and updates a major master file and interfaces with another major system.
7-8	35-60 programs or 20-100 man months of maintenance or 1,000-10,000 computer hours annually and updates a master file and interfaces with another system.
5-6	10-34 programs or 10-20 man months of maintenance or 250-5000 computer hours annually.
3-4	5-9 programs or 5-9 man months of maintenance or 50-249 computer hours annually.

2 and below other system

**The Company Assets**

9-10	Directly affect cash
8	Affects movement of assets
6-7	Indirectly affects movement of assets
5 and below less affect on assets	

**The Critical System Index**

9-10	Necessary to maintain daily business
7-8	Necessary to maintain statutory requirements and monthly reporting
5-6	Necessary to maintain business
4 and below not primary to business	

Figure 5.

**DEGREE OF LOSS MATRIX**

HAZARD:		RATINGS	
NO.	TYPE	LOSS	FREQ.
DEGREE OF LOSS:	MANIFESTATIONS:		
A.	MINOR ANNOYANCE		
B.	MAJOR ANNOYANCE		
C.	MINOR LOSS/RECOVERY		
D.	MAJOR LOSS/RECOVERY		
E.	MAJOR INTERRUPTION		
F.	SEVERE DISRUPTION		
G.	MAJOR DISASTER		

Figure 6.

*Example 4*

Objective: Modify Risk Analysis procedure to permit an evaluation of risks that do not lend themselves to monetary measurement criteria such as events involving adverse social or political consequences—This organization is experimenting with the coupling of conventional Risk Evaluation Procedures with a unique “sensitivity value” or point scale (Figure 7), in order to measure critical events which do not permit monetary assignments.

*Example 5*

Objective: More clearly distinguish between “major” and “minor” threats and classes of exposure present in an information system—In the interest of simplifying the Risk Analysis procedure and at the same time focus attention on the threats of potentially great severity, this very large computer user developed a unique two-level threat classification system (Figure 8). They also divided potential security exposures into four categories. (Figure 9).

**CONCLUSION**

It is very difficult to prove that a computer system Risk Analysis study will necessarily result in a more secure in-

**SENSITIVITY VALUE SCALE**

Sensitivity Value Factor plus Back-up Factor may be used to calculate "Exposure Points Value" per year --

<b>Example</b>	
Asset List	Value Points
<ol style="list-style-type: none"> <li>1. Operators manual</li> <li>2. System reference manuals</li> <li>3. Operational files</li> <li>4. Data Base file</li> <li>5. Program Library</li> </ol>	<p>10</p> <p>50</p> <p>100</p> <p>250</p> <p>300</p>
⋮	⋮
ETC.	ETC.

Figure 7.

**MAJOR/MINOR THREAT CATEGORIES**

<b>MAJOR THREATS:</b>	An event of catastrophic proportions which destroys the ADP facility or renders it inoperable. Examples: fire, flood, earthquake, tornado, bombing, riot. Assumption is made that all attendant areas of the facility, such as the tape/disk library, are destroyed. Relocation to an alternate processing site is required. Only the material stored off-site is available for use.
<b>MINOR THREATS:</b>	This category includes all the failures, errors, and mishaps encountered daily. While each occurrence may result in relatively short processing delay or minor distortion or loss of data, the cumulative cost of many occurrences can be significant. Examples: CPU failure, wrong tape or pack mounted, listings lost, air conditioning failure.

Figure 8.



SECURITY EXPOSURE IMPACT CLASSIFICATION	
Security Exposure	Possible Results of Security Failure
1. Data Integrity	Destruction or unauthorized modification of data, unintentional or deliberate.
2. Data Confidentiality	Unauthorized disclosure of sensitive data.
3. Operational Reliability	Undependable or inadequate processing; unavailability of processing. (Processing should be accurate, dependable, and timely.)
4. Asset Integrity	Destruction or physical damage to buildings and equipment and supporting functions.

In general, the first three categories represent threats to data and processing. Asset integrity can most often be related to physical assets: equipment, supplies, furniture, storage media, etc.

Figure 9.

formation system. On the other hand, as evidenced by the survey covered in this paper, computer users need a systematic way to study, evaluate and prioritize the risks and countermeasures associated with their systems. Risk Analysis lends itself to this task.

Fortunately, there are many risk investigation methods from which to choose. Different organizations should use the methodology that gets the job done, at a price they can

afford to pay. There is no question that data processing users need to become more knowledgeable regarding their system vulnerabilities and risk prevention methods. Therefore, slowly but surely, we will probably see more organizations implementing a risk or threat or vulnerability analysis in one form or another. The procedure used will not be nearly as important as the overall results that will be obtained.

## REFERENCES

1. ADP Security Handbook—Handbook Supplement: DIPS Manual Chapter 6, U.S. Department of Agriculture, 1977.
2. Carrol, John M., *Computer Security*, Security World Publishing Company, Inc., 1977.
3. *Computer Security Risk Analysis and Control: A Guide for the DP Manager*—National Computing Centre, Ltd., 1979 (available in the U.S. from Hayden Book Company, Rochelle Park, N.J. 07662).
4. Courtney, R. H., "Security Risk Assessment In Electronic Data Processing Systems," IBM Corp., 1975.
5. "Guidelines for Automatic Data Processing Physical Security and Risk Management," FIPS Publication 31, National Bureau of Standards, June 1974.
6. Koenig, Rick, "How to Get a System Security Project Off the Ground!" *Computer Security and Privacy Symposium Proceedings*, Honeywell Information Systems, 1977.
7. Kraus, Leonard S. and MacGahan, *Computer Fraud and Countermeasures*, Prentice Hall, 1979.
8. Martin, James, *Security, Accuracy, and Privacy On Computer Systems*, Prentice Hall, 1973.

# A mathematical model of character string manipulation

by SAKTI PRAMANIK

Indiana University—Purdue University at Indianapolis,  
Indianapolis, Indiana

## INTRODUCTION

Every insert and delete operation on a string of characters causes it to expand or contract in memory. If on the other hand, these commands are saved, the character string can be updated at the end of the editing session.<sup>1,2</sup> The advantage of doing this is to move any character at the most once in memory. This is possible because the final position of a character in the string can be determined from the saved edit functions. In Figure 1 below it is shown that a "Delete character in position 3" and then an "Insert a character  $X$  before the 3rd character position" can be combined into a single edit function, "Replace the character in position 3 by the character  $X$ ."

### Delete and insert functions

Initial Character string	After Delete	After Insert
A B C D E ↙ Delete	A B D E ↙ Insert X	A B X D E

### Combining the edit functions

Initial character string	After Replacement
A B C D E ↙ Replace by an X	A B X D E

Figure 1—Combining the effect of delete and insert functions into a single edit function.

One difficulty of combining several edit functions as above is that they are issued relative to the character position in the current string while the combined edit function has to work on the characters in the initial string. For example, deleting the  $i$ th character of the initial string and then deleting the  $i$ th character of this updated string mean deleting the  $i$ th and the  $(i+1)$ th characters of the initial string. It seems that the above combining process requires the transformation of the editorial point in the current string into a corresponding editorial point in the initial string. But this conversion may not always be possible because the mapping may not exist.

Instead of defining the editorial point on a reference frame which is always changing a static frame of reference is created. This is done by defining a sequence of character slots. Characters of the string are stored in these slots in sequence. Three microscopic edit operators are defined which operate on the contents of these slots rather than on the character direct by its position in the string.

## DEFINITION OF THE OPERATORS AND THEIR ALGEBRAIC PROPERTIES

Let  $S_i$  denote the  $i$ th character slot. So  $S_i$  also represents the slot location of the  $i$ th character in the current string. Thus,  $S_1$  denotes the slot containing the left most character in the string. The three basic micro-operators,  $B_i^X$ ,  $F_i^X$ , and  $R_i^X$ , are then defined as follows:

$$B_i^X \equiv S_i \leftarrow 'X'$$

i.e., the character  $X$  is moved into the  $i$ th slot and the previous character in that slot is lost.

$$F_i^X \equiv B_{i+1}^X, S_{i+1} \leftarrow C(S_i)$$

i.e., the character in the  $i$ th slot is moved into the  $(i+1)$ th slot and the previous character in that  $(i+1)$ th slot is lost. Then the character  $X$  is moved into the  $i$ th slot. Similarly,

$$R_i^X \equiv B_{i+1}^X, S_i \leftarrow C(S_{i+1})$$

the content of the  $(i+1)$ th slot is moved into the  $i$ th slot and the previous character in the  $i$ th slot is lost. A character  $X$  is then moved into the  $(i+1)$ th slot.

In the above definition,  $X$  represents any single character; for blank characters however, no superscript will be used. For example,  $B_i$  will move a 'blank' character into the  $i$ th slot. The following equivalence relations hold good for the above operators. They indicate that the operators on the right hand side of a relation eventually result in the same characters in the slots as those operators on the left hand side; but the ones on the right hand side achieve this by a minimum amount of character movement. The convention that the operators in a string gets executed in sequence from right to left, is assumed. For example, in the operator string  $B_i^X F_i^Y, F_i^Y$  is executed first, and then  $B_i^X$ .

$$F_i^X R_i^Y \equiv B_i^X \quad (1)$$

$$F_{i+1}^X R_i^Y \equiv B_{i+2}^Y R_i^X \quad (2)$$

$$F_i^X R_{i+1}^Y \equiv B_{i+2}^Y F_i^X \quad (3)$$

$$R_i^X F_i^Y \equiv B_{i+1}^X \quad (4)$$

$$R_{i+1}^X F_i^Y \equiv B_i^Y R_{i+1}^X \quad (5)$$

$$R_i^X F_{i+1}^Y \equiv B_i^Y F_{i+1}^X \quad (6)$$

$$F_i^X B_i^Y \equiv B_i^X B_{i+1}^Y \quad (7)$$

$$F_i^X B_{i+1}^Y \equiv F_i^X \quad (8)$$

$$B_i^X F_i^Y \equiv F_i^X \quad (9)$$

$$B_{i+1}^X F_i^Y \equiv B_{i+1}^X B_i^Y \quad (10)$$

$$R_i^X B_i^Y \equiv R_i^X \quad (11)$$

$$R_i^X B_{i+1}^Y \equiv B_i^Y B_{i+1}^X \quad (12)$$

$$B_i^X R_i^Y \equiv B_i^X B_{i+1}^Y \quad (13)$$

$$B_{i+1}^X R_i^Y \equiv R_i^X \quad (14)$$

The above operators commute under the following conditions:

$F_i^X$  commutes with  $F_j^Y$ ,  $R_i^X$  commutes with  $R_j^Y$ , and  $F_i^X$  commutes with  $R_j^Y$  only when  $|i-j| > 1$ .  $F_i^X$  commutes with  $B_j^Y$ , and  $R_i^X$  commutes with  $B_j^Y$ , only when  $j < i$  or  $j > i+1$ .

#### DEFINITION OF INSERT AND DELETE FUNCTIONS

Function to insert a character  $X$  before the  $i$ th character in a character string of length  $N$  is represented by the operator string

$$F_i^X F_{i+1}^X \dots F_{N-1}^X F_N^X$$

where the  $F$ s are performed in a sequence from right to left. So  $F_N^X$  moves the character in the  $N$ th slot of the  $(N+1)$ th slot and then moves the character  $X$  into the  $N$ th slot. Then  $F_{N-1}^X$  moves the character in the  $(N-1)$ th slot to  $N$ th slot and then the character  $X$  into the  $(N-1)$ th slot, and so on.

The function to delete the  $i$ th character in a character string of length  $N$  is represented as follows:

$$R_{N-1} R_{N-2} \dots R_{i+1} R_i$$

Similarly, the  $R$ s are performed from right to left in the operator string above. We can now represent a sequence of insert and delete functions by concatenating the strings of  $F$ s and  $R$ s. The temporal sequence of the edit functions are preserved by concatenating the operator string of the next edit function to the left of the string of the previous edit functions. The advantages of representing a sequence of edit functions by a concatenated operator string is that this string can be simplified considerably by using the relations 1 through 14. For example, the delete and the insert functions of Figure 1 is represented by the operator strings  $R_4 R_3$  and  $F_3^X F_4^X$  respectively. To represent the combined effect of

the two edit functions, we concatenate the operator strings as follows:

$$F_3^X F_4^X R_4 R_3$$

By using relation 1,  $F_4^X R_4$  is reduced to  $B_4^X$ . This  $B_4^X$  in turn can be merged with  $F_3^X$ . The above operator string now becomes  $F_3^X R_3$  which by using Relation 1 again becomes  $B_3^X$ . Thus the combined effect of the two edit functions is the micro-operator  $B_3^X$  which imply moving the character  $X$  into slot 3. It should be noted that slot 3 is also the position of the third character in the initial character-string. In other words,  $B_3^X$  replaces the third character in the initial characterstring by a character  $X$ . A systematic approach to combine a sequence of edit functions to produce an optimized operator string is presented in the following section.

#### REDUCTION ALGORITHM

This algorithm starts by merging the operator strings of the first and the second edit functions. The string thus obtained is now merged with the operator string of the third edit function. This process of merging two operator strings at a time continues until the operator strings of all the edit functions have been merged. The merging is done by taking one operator from the left string and combing toward right through the operators of the right string until it combines with an operator or finds a place in the right string where it can not be combed any further to the right. The combing is essentially done by commuting the operator successively with its right neighbor in the string. Combining two operators involves finding the previous relation whose left hand side corresponds to these two operators; then replacing them by the operators of the right hand side of the relation. These operators may now be combined with other operators of the right string. The following example shows how the operator strings for the three edit functions have been merged into a single operator string. The edit functions, in the order they are issued, are: (1) insert a character  $X$  before the 9th character; (2) insert a character  $Y$  before the 9th character; (3) delete the 6th character.

Assuming an initial character string of length 10, the operator strings for the above edit functions are: (1)  $F_9^X F_{10}^X$ ; (2)  $F_9^Y F_{10}^Y F_{11}^Y$ ; (3)  $R_{11} R_{10} R_9 R_8 R_7 R_6$ ; respectively.

The merging process starts with the operator string of the first edit function, i.e.,  $F_9^X F_{10}^X$ . The second operator string is now merged with this from left as follows:

$$F_9^Y F_{10}^Y F_{11}^Y \xrightarrow{\text{merge with}} F_9^X F_{10}^X$$

$$\text{Producing } F_9^Y F_{10}^Y F_9^X F_{11}^Y F_{10}^X$$

See that  $F_{11}^Y$  of the left string has commuted with  $F_9^X$  of the right string. This resulting string is now merged with the 3rd operator string as follows:

$$R_{11} R_{10} R_9 R_8 R_7 R_6 \xrightarrow{\text{merge with}} F_9^Y F_{10}^Y F_9^X F_{11}^Y F_{10}^X$$

$R_6$  combs all the way through the right string and so does

$R_7$  producing

$$R_{11}R_{10}R_9R_8 \xrightarrow{\text{merge with}} F_9^Y F_{10}^Y F_9^X F_{11}^Y F_{10}^X R_7 R_6$$

$R_8$  now combines with  $F_9^Y$ , thus both of them are replaced by  $B_8^Y F_9$ , as follows:

$$R_{11}R_{10}R_9 \xrightarrow{\text{merge with}} B_8^Y F_9 F_{10}^Y F_9^X F_{11}^Y F_{10}^X R_7 R_6$$

The new operator  $B_8^Y$  is now combed through its right neighbors and eventually combines with  $R_7$  to produce

$$R_{11}R_{10}R_9 \xrightarrow{\text{merge with}} F_9 F_{10}^Y F_9^X F_{11}^Y F_{10}^X R_7^Y R_6$$

Now  $R_9$  combines with  $F_9$  to produce  $B_{10}$ , and this  $B_{10}$  combines with  $F_{10}^Y$  to produce  $F_{10}$ . Continuing this process we get the final merged string

$$F_9^X F_{10}^X R_7^Y R_6$$

The resulting merged string consists of a disjoint set of substrings of only  $F$ s, or only  $R$ s, or only  $B$ s. For example, the merged string above consists of the disjoint substrings  $F_9^X F_{10}^X$  and  $R_7^Y R_6$ . These two substrings are disjoint because  $R_7^Y R_6 F_9^X F_{10}^X$  is equivalent to  $F_9^X F_{10}^X R_7^Y R_6$ . The relative order of these disjoint substrings is important, however, for the merging algorithm discussed above because it may result in an incomplete merge if the substrings are not properly ordered. For example, a fourth edit function "Delete the 8th character" is issued after the three edit functions, discussed above. The operator string for the fourth edit function is  $R_{10}R_9R_8$ . If the right string is  $R_7^Y R_6 F_9^X F_{10}^X$  then the merged string will be  $R_{10}R_9R_8R_7^Y R_6 F_9^X F_{10}^X$ . This cannot be reduced any further because  $R_8$  does not commute with  $R_7^Y$ . On the other hand if the right string is  $F_9^X F_{10}^X R_7^Y R_6$  then the merged string is reduced considerably to  $R_7^X R_6$ . This is because  $R_8$  combines with  $F_9^X$ , and so on.

It can be shown that the merging algorithm will always produce a completely reduced string when the disjoint substrings of the right string are kept in descending order from left to right; the ordering is done by their highest subscript.

A computer program has been written to implement the above merging algorithm. For random input data it is found that the number of commutations (required for combing) increases very rapidly with the increasing length of the character string. The number of commutations can be reduced considerably if combing is done for a group of operators at a time rather than taking only one operator at a time from the left string. This reduction is possible because the subscripts of the operators within a group are sequential. The following table gives the merged substrings in terms of the

TABLE I.

	$i = j$	$i > j$	$i < j$
$1 = k$	$B_{k+1}^Y$	$B_{i+1}^Y B_i^X - 1 R_i^Y - 2; j$	$B_{k+1}^Y F_{i; j-2}^X$
$1 < k$	$F_{i+1}^Y + 1 F_{1+2}^X; k$	$R_{i-2}^X R_{i-3}^Y; j F_{i+1}^Y F_{1+2}^X; k$	$F_{i; j-2}^X F_{i+1}^Y F_{1+2}^X; k$
$1 > k$	$R_{i; k+1}^Y$	$R_{i; k+1}^X R_{i-2}^Y R_{i-3}^Y; j$	$R_{i; k+1}^Y F_{i; j-2}^X$

subscripts of the merging substrings. We will use the following shorthand notation for the substrings:

$$F_{i;j}^X = F_i^X F_{i+1}^X \dots F_j^X \text{ for } j \geq i \\ = \text{Null For } j < i$$

and

$$R_{i;j}^X = R_i^X R_{i-1}^X \dots R_j^X \text{ for } j \leq i \\ = \text{Null For } j > i$$

The entries in Table I show the result of merging the substrings  $R_{1;j}^Y$  from left with  $F_{i;k}^X$ . Similar table can be constructed for merging  $F_{i;k}^X$  from left with  $R_{1;j}^Y$ .

CONCLUSION

The merged operator string works as an efficient mapping between the original character string and the updated character string. For example, a text file on a tape unit can be considered as a continuous string of characters<sup>3</sup> and all the update information about this text string can be maintained through a merged operator string. The merging process requires a fair amount of string manipulation. This, however, remains bounded when the number of inserts and deletes are evenly distributed over time and space.

ACKNOWLEDGMENT

I would like to thank professors Edgar T. Irons and Alan J. Perlis of Yale University for many helpful conversations.

REFERENCES

1. Pramanik, Sakti, "Map Editing," Ph.D. thesis, Yale University, 1974.
2. Pramanik, Sakti and Irons, Edgar T., "A Data-Handling Mechanics of On-Line Text Editing Systems with Efficient Secondary Storage Access," *Proceedings of National Computer Conference*, 1979.
3. Wilkes, A., "An On-Line Algorithm for Manipulating long Character Strings," *IEEE Transactions on Computer*, November, 1970.



# Policy, values and EFT research: anatomy of a research agenda

by KENNETH L. KRAEMER

University of California  
Irvine, California

and

KENT W. COLTON

Brigham Young University  
Provo, Utah

## INTRODUCTION

In its Final Report of February 1977, the National Commission on Electronic Fund Transfers (NCEFT) defined EFT as:

“... a payments system in which the processing and communications necessary to effect economic exchange, and the processing and communications necessary for the production and distribution of services incidental or related to economic exchange, are dependent wholly or in large part on the use of electrons.”<sup>36</sup>

This innocuous definition of EFT hardly fits with the emerging recognition that EFT is a technologically-based system with the potential for vastly changing relationships among private enterprises, public institutions and individuals throughout the country.<sup>2,15,28,27</sup> The complexity of EFT is illustrated by the fact that it is not a single technological application; nor is it even composed of a unified group of technological applications. At least five different techniques characterize the applications being developed in this country: pre-authorization procedures, automated banking through EFT terminals; point-of-sale (POS) operations; national bank card networks; and automated clearinghouse procedures (Table I). Individually and through a combination of these subsystems and techniques, EFT operating systems are being established in various areas throughout the country.<sup>20,34,35,36,40,43</sup> However, such efforts are not established in a vacuum; their success, failure, and very nature are highly dependent on several major forces surrounding their development.

### *The evolution of EFT: major forces and values*

Figure 1 presents an overview of the major forces involved in the evolution and development of EFT systems: institutional actors, EFT technology and operating systems, EFT

regulation and control, impacts of EFT on people and the economy, and monitoring and evaluating of EFT systems. Thus, the actual EFT operating systems are only one part of a much larger system. And because EFT operating systems are integrally linked with these other forces, they inherently are involved in the major public policy and political questions which traditionally relate to these other forces.

To begin, the *institutional actors* involved in the provision and use of EFT technology play an important role in determining the overall shape of EFT operating systems. *Consumers* or *users* of EFT technology include those individuals, businesses or governments who currently use EFT or who may do so in the future. What consumers and users will accept is a major determinant of what the *providers* of EFT, the financial institutions and retailers who currently offer or who could potentially offer EFT services, will offer. But the providers also may be expected to induce consumers to accept services that are unfamiliar and, perhaps, not always in the consumers' interest. The providers of EFT also must deal with the *suppliers* of the technology, those who actually produce the hardware and software and supply specific services and equipment. Generally, the interests of these two groups—providers and suppliers—tend toward promotion and rapid deployment of EFT technologies.

Standing between these actors and EFT technologies are the government agencies which regulate and control EFT systems both directly and indirectly through regulation of the basic interactions among providers, suppliers, consumers and users. State and federal laws and regulations not only prescribe the extent and nature of EFT development, but often set the framework within which EFT innovation may occur. Thus, the various *institutional actors* will seek to influence regulation in their favor, and in turn, regulation and control will set boundaries for the development of EFT technology.

Once developed, *EFT operating systems and technologies* will have a substantial impact on society—*impacts on people* and *impacts on the economy*. These impacts may be ex-

TABLE I.—Major EFT Applications Currently in Development

Subsystem or technique	Operation
<u>Preauthorization Procedures</u>	
- Direct deposit of regular payments (such as paychecks, welfare payments, retirement checks, stock dividends).	Once authorized, such deposits and payments made automatically and electronically according to agreed-upon procedures.
- Direct payment of recurrent expenses (such as house and car payments, utilities).	
- Telephone "bill-payment."	Customers use the telephone to authorize financial institutions to pay monthly bills (generally through electronic transfers) or to transfer money from a savings to a checking account or vice versa.
<u>Automated Banking through EFT Terminals</u>	
- Use of Automatic Teller Machines (ATMs) to automate traditional banking activities such as depositing and withdrawing money from accounts or cashing checks.	ATMs provides 24-hour banking service through electronic terminals; almost 8,000 ATMs in place today.
- Authorization of credit and checks.	A terminal is used to check the customer's credit and to determine whether the checking or credit accounts have adequate funds to handle the transaction in question.
<u>Point of Sale (POS) Operations</u>	
- Facilitation of electronic transfer of money at the point of actual operation or sale (with a direct link to the customer's account).	Verify or guarantee a check electronically, or make a direct, electronic debit from a purchaser's account to the account of a business establishment at the point of sale (e.g., the "debit card").
<u>National Bank Card Networks</u>	
- Clearing of credit card vouchers.	Facilitate the electronic exchange of credit transactions and vouchers (e.g., National Bank Americard, Inc., Interbank Card Association).
<u>Automated Clearinghouse (ACH) Procedures</u>	
- Facilitation of electronic exchange of money (both debits and credits) among financial institutions.	An electronic network(s) substitutes for the paper-oriented check-clearing system. Also provides the clearing facility for preauthorization procedures and POS operations, but use is small.

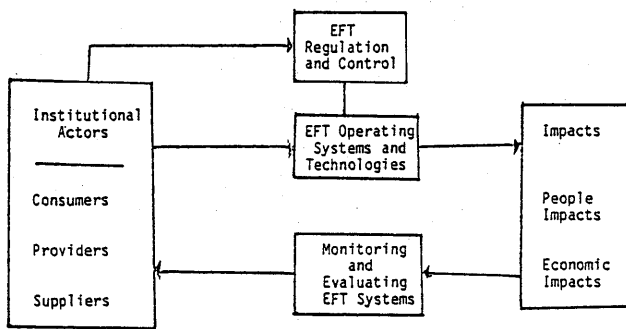


Figure 1—The dynamics of EFT evolution and development.

pected to reinforce or to change any or all of the actors' views of the desirability of further developments. Indeed the evaluation and assessment of these impacts provides an essential "feedback loop" for understanding the dynamics of EFT development and a means of *monitoring and evaluating EFT systems* from the standpoint of each actor's interest. Given the varying interest, it is inevitable that conflict will develop among them and need to be resolved through the policy process.

The perspective and values which an individual brings to this framework will naturally result in varying assessments of the shape such impacts will take and the relevant parties to be included in the policy making process. Kling [29, p. 643] has identified five major value orientations implicit in the published discussions of EFT systems:

#### Private enterprise model

The pre-eminent consideration is profitability of the EFT systems, with the highest social good being the profitability of the firms providing or utilizing the systems. Other social goods such as users privacy or the need of the government for data are secondary.

#### Statist model

The strength and efficiency of government institutions is the highest goal. Government needs for access to personal data on citizens and needs for mechanisms to enforce obligations to the state always prevail over other considerations.

#### Libertarian model

The civil liberties as specified by the U.S. Bill of Rights are to be maximized in any social choice. Other social purposes such as profitability or welfare of the state are to be sacrificed if they conflict with the prerogatives of the individual.

#### Neopopulist model

The practices of public agencies and private enterprises should be easily intelligible to ordinary citizens and be re-

sponsible to their needs. Societal institutions should emphasize serving the common man.

#### Systems model

The main goal is that EFT systems be technically well organized, efficient, reliable, and esthetically pleasing.

#### Developing an agenda for EFT research

We maintain that the value orientations which characterize the development of EFT also characterize the development of a research agenda for EFT. For example, when Congress established the time frame for the NCEFT, they insured (consciously or unconsciously) that both the relevant policy questions and the answers would be defined largely by the current participants in the U.S. payments system—the financial and retail community. This occurred because the short time frame meant that new research could not be conducted, and that the Commission would have to rely on information mainly in the hands of financial institutions, their technology suppliers, and their consultants.<sup>5</sup> Of course, that fact also increased the opportunity for these actors to decide what research would be reported and what would not be reported.

This paper will describe a project to frame a research agenda *broader* in scope than the efforts of the NCEFT. Supported by a grant from by The National Science Foundation, we developed a process\* which brought together representatives of not only the key current participants but also representatives of other actors who are not now but who might become participants in the future—representatives of the full set of actors implied in Figure 1. These included representatives of: financial institutions such as banks and savings and loans; non-depository institutions such as department stores and supermarkets; state government institutions such as EFT commissions and regulatory agencies; federal government users such as the operators of automated clearing houses and government regulators such as the Federal Reserve Board and the Federal Home Loan

\* The authors of this paper were the principal investigators of this project. A two-part process was used. It included an exploratory workshop to initially review and identify research needs, and a follow-up conference to more fully analyze ideas and to develop an agenda for future research. [8, Chapter 1] The exploratory workshop was conducted in Washington, D. C. on November 18 and 19, 1976, and the Conference was held in Boston on June 2 and 3, 1977. Because the Conference presented a unique opportunity to systematically solicit the opinions of a broad cross section of interests in EFT, a questionnaire was circulated to all 94 participants seeking to determine their opinions on 37 research issues (culled from the 60 suggestions in the exploratory workshop). The participants were asked to first rank the ten most important from the perspective of national interest, and which were most important from the perspective of the respondent's institutional interests. The aim of this data collection exercise was not only to derive priority ranking for research issues, but to develop priority rankings for different "target populations."

The agenda-making process was planned and organized by Public Systems Evaluation, Inc. (PSE), of Cambridge, Massachusetts, and the Public Policy Research Organization (PPRO) of the University of California at Irvine. They were supported by a grant from the National Science Foundation to Public Systems Evaluation, Inc.



Bank Board; consumers including national consumer organizations and individual consumers; and the EFT technology industry such as equipment vendors and consultants. Moreover, we asked these people to consider what research was needed in the middle and the longer term, and whether the research was primarily important from their institutional interest, or from a broader national interest or both.

We present the agenda which resulted from this process both for its inherent value in informing future research, and for its value in illustrating the conflicting value orientations which characterize the field. What emerges from our analysis is a clear indication that the very definition of what should be examined by the scientific method involves value conflicts not only because of the answers which might result, but also because the simple act of including an issue on the agenda might have political consequences. For example, both the providers and the suppliers of EFT technology continually expressed concern over research into potentially harmful impacts of EFT, such as consumer abuse, consumer costs and consumer problems. They were worried that the very existence of such research might be construed by some policy makers as reason for slowing the development of EFT.<sup>7</sup>

Thus, in the agenda itself and throughout its making, it is apparent that different actors prioritize and approach key issues differently depending upon their institutional setting and values. Although other value differences are apparent, the most consistent tension was between those who favored free market determination of how EFT develops and those who favored more systematic, deliberate consideration of what societal goals should be served by EFT, given a fuller understanding of the likely and potential uses and impacts. The former perspective, based on the *private enterprise model*, sees market research conducted by the financial institutions in their own enlightened self-interest as the dominant if not the sole kind of research that is needed. Moreover, public acceptance and the use of EFT systems is viewed as proof of the appropriateness of this perspective, regardless of whether certain consumer groups are excluded. The latter perspective, which favored more systematic consideration of societal goals, is based on the more socially-oriented *libertarian* and *populist models*. It questions the adequacy of industry-dominated market research not only for the protection of consumers but also for the exploitation of the technology's potential in the broader interest of society. The following exchange illustrates these varying perspectives:

Cox: I think that we are witnessing a very fascinating series of statements here in which some of us who would really like to challenge the fundamental assumptions on which our political system, our economic system, and maybe even the religious basis of our society rest, have decided to use EFT as the vehicle for it today. And I think that makes it very hard to discuss EFT. It is not a vital issue on which the fate of our culture turns, but it is a matter about which we ought to let the free and most open conditions that we can create lead us to whatever mix of wampum and currency and electronic signals that we like be the means by which we carry on the business of our society.

Coates: You might easily get the impression that I am opposed to EFTS. In fact, I'm very enthusiastic about it . . . But . . . there are very serious structural problems that are best addressed now rather than later . . . to leave the problem of its (EFTS) development solely in terms of incremental change and specific cases and issues in an allegedly free enterprise evolutionary process is the exact analog of treating the problems of women workers in society on an individual piecemeal basis. To overlook the fundamental structural elements is effectively to do a systematic injustice. What you take to be dragging in all the dirty linen of society as part of the question is absolutely essential. But it is not an ideological move peculiar to EFTS. It is rather something that has to be done with all our major technological systems.<sup>7</sup>

It is precisely this debate, which mirrors the larger debate in the society, to which this agenda for EFT research is addressed.

## AN AGENDA FOR EFT RESEARCH

Table II presents the EFT research agenda in outline form as a guide to the discussion which follows. Interestingly, this agenda relates closely to the major forces of EFT development outlined in Figure 1. It is significant to note that within the context of the five major areas used to organize the agenda-making process, twelve research topics have been highlighted as warranting first priority attention. The agenda is not a list of specific research projects that need to be conducted, but rather a listing and discussion of research areas in which specific projects might be formulated. Hopefully, it will serve to help stimulate specific research projects.

The areas given priority on the agenda correspond with responses to a survey questionnaire about EFT research issues. Respondents were asked to identify, from a list of 37

TABLE II.—EFT Research Agenda

I. TECHNOLOGICAL ISSUES IN EFT	Clarification of Technological Issues Warranting Research
II. EFT IMPACTS ON PEOPLE	Costs and Benefits of EFT to the Consumer
	Consequences of EFT for the Less Advantaged
	Education of Consumers for Dealing with EFT
	Records Control and Consumer Protection under EFT
	Effective and Acceptable Privacy Safeguards
III. ECONOMIC IMPACT OF EFT	Costs of the Current Financial Payments System
	Impact of EFT on the Financial System and Financial Institutions
IV. REGULATION AND CONTROL OF EFT	Identification and Understanding of New Regulatory Issues Arising from EFT (including the questions: Is regulation needed at all?)
	Study of the Range and Options of Organizing EFT-Related Regulatory Structures
V. EVALUATING AND MONITORING EFT SYSTEMS	Long-Run Interactions between EFT and Society
	EFT as a Study of Technological Change and Impact

issues, the ten issues they believed of highest priority, and then to rank those issues in descending order of importance (with 10 points for the most important, 9 points for the second most important, etc.). In addition, an "intensity" score was calculated by dividing the number of times the issue was ranked in the top ten by the total points it received. (In other words, if the research issue area has an intensity of 7, it means that on the average those who ranked this issue in the top ten felt it was the third most important issue.) Rankings of intensity are shown in Table III. Finally, respondents to the survey were asked to characterize their institutional view depending on whether they represented a provider, supplier, regulator, researcher, or "other" perspective. And as noted earlier, in a number of cases the ranking of agenda items varied according to these institutional perspectives.\*\*

### *Technological issues in EFT*

None of the technological issues in EFT ranked high among research topics (Table III). However, the research dimensions of this topic require additional attention because the technologists take three different and conflicting positions on the issues and have tended to prevent clarification.

The first position is that there are no serious technological issues in EFT. The technology can do whatever people want it to do for EFT, and adequate safeguards can be provided for potential EFT technology problems, whether arising from the technology *per se* or from its use. Those whose values hold close to the *systems model* generally take this view, and the only issue is whether people are willing to pay for adequate safeguards. Therefore, the question is political and economic, not technological. The second position, found particularly among some technology providers, is that there *probably* are no serious technological issues, but the technologists cannot talk about the issues in detail because of possible implications for Federal Communications Commission hearings, government and private litigation or industry competition. The third position, found among some academicians versed in technology, generally those more tied to *libertarian and populist value models*, is that there really are serious issues regarding reliability, security standards and competition. "Somebody," they argue, should look into these issues, although they, personally, are not deeply interested in the problem.

The result of this standoff from engagement with technological issues is that the issues continue to receive inadequate public definition and research attention. Conse-

\*\* The listing of perspectives noted above represents a collapsing of categories. Respondents were actually asked to classify themselves according to narrower categories, and these were grouped as follows: *providers*: financial, non-depository institutions; *regulators*: state government, federal government; *supplier*: EFT technology industry, other industry; *university researchers*: business/economics departments, other university departments. No special category was provided for "consumers" *per se* since all of the groups noted above are also consumers. There were several people at the conference who represented specific national consumer groups, and their responses are included in the "other" category.

quently, we believe that additional effort must be focused on defining issues requiring research and bringing together representatives of the varying technological perspectives to confront one another directly. Furthermore, research should be conducted on the three issues which received the highest ranking among technological concerns. The first is technical security of EFT systems. EFT development is proceeding without adequate solutions to computer and telecommunications security, without standards for appropriate levels of security, and without any universally acceptable solution in sight. The second issue is the appropriate configuration of EFT networks. The number of EFT delivery systems is expected to increase from its present 200 to perhaps 500 independent systems in the 1980's. Yet, in later decades as the economies of scale become apparent, these systems are expected to be integrated into perhaps only 100 large networks.<sup>19</sup> Some systems undoubtedly will be patched together whereas others will be totally rebuilt. Patched systems are predicted to be security risks and prohibitively costly. The integration of EFT networks raises problems of reliability, security, efficiency and standardization which will need to be solved in order to link existing systems and/or to build new integrated systems.<sup>38,42</sup> The third issue is the future interface of general purpose telecommunications capabilities and EFT. The future interface of EFT systems with general-purpose telecommunications capabilities potentially extends the scale of networking problems immensely beyond any systems currently in existence.<sup>51</sup> It also raises new problems regarding priority, consolidation of transaction data and international protocol, among others.

### *EFT impacts on people*

The impacts of EFT on people, particularly consumers, generally received the highest ranking of all research. The costs and benefits of EFT to the consumer was the highest ranked individual topic both in terms of total points and in terms of highest intensity rating (Table III). In addition, this issue was ranked relatively high for almost all institutional sectors. The data from the survey displayed in Table IV reflects this broad base of support, with all but one institutional sector ranking this consumer related issue with an intensity of seven or more.

Despite this general agreement on the importance of consumer issues, there was disagreement about how to insure adequate attention to them. Some conference participants felt that market research and competition among financial institutions—the *private enterprise model*—would insure that consumer interests are adequately met by EFT developments. However, others were less sanguine about this prospect. The following exchange illustrates this concern:

Coates: As major consultants to the industry, could you tell us what kinds of socially conscious questions come forward from your clients?

Horan: I think that all market research, all planning of banks,

TABLE III.—Ranking of Research Issues by Conference Participants

Issue	Total Points	Rank among the 37 Issues	Intensity
TECHNOLOGICAL ISSUES IN EFT			
Security of EFT	86	15	5.1
Alternative network approaches	84	16	4.9
Alternative communication systems and EFT	82	18	4.8
Reliability of EFT	72	23	4.5
Implications of potential	70	24	4.4
Reversibility of EFT changes	68	25	5.7
EFT IMPACTS ON PEOPLE			
Costs and benefits of EFT to the consumer	269	1	7.3
Educating consumers regarding their EFT-related rights	149	5	5.5
EFT impact on low income consumers	130	8	5.4
EFT impact on individual surveillance	95	14	6.3
Consumer abuse (debit cards stolen, payments initiated)	82	19	4.3
Impact of EFT on consumer behavior	81	20	4.3
Record controls and counterfeiting under EFT	53	28	4.1
Privacy problems and EFT	40	33	4.4
Ombudsman as a means of consumer protection	31	35	5.2
Effects of mandatory disclosure laws on consumer	10	37	3.3
ECONOMIC IMPACT OF EFT			
Comparative costs of current payments system and EFT	268	2	6.2
Impact of EFT on market competition	104	12	4.2
Impact of EFT on the definition and velocity of money	83	17	4.2
Impact of EFT on smaller financial institutions	67	26	3.7
Impact of EFT on operating and other expenses	63	27	3.3
EFT and float	45	30	2.8
EFT-induced changes in monetary systems	41	32	4.6
Impact as a result of EFT fraud	22	36	5.5
REGULATION AND CONTROL OF EFT			
Definition of EFT regulation; what should be regulated	202	3	6.1
Access rules for EFT data	143	6	5.7
Federal government as operator of EFT systems	140	7	5.4
Need for EFT regulation	115	11	5.8
Roles of various federal and state legislatures/regulators	100	13	5.6
Institutionalizing consumer interests	76	21	4.5
Equal access of all to EFT	73	22	4.6
What private institutions should be regulated (bank, nonbank, etc.)?	51	22	3.6

TABLE III. (Continued)

Issue	Total Points	Rank among the 37 Issues	Intensity
EVALUATING AND MONITORING EFT SYSTEMS			
EFT impact on the long-range character of society	199	4	6.6
Impact of EFT on other societal institutions	127	9	5.8
EFT as a case study of technological change and impact	116	10	3.7
EFT development in other countries	44	31	4.0
EFT and international fund flows	32	34	4.6

of merchants, of depository institutions are premised on the fact that if the consumer doesn't accept it, you just don't have a market. It's a commercial type of decision.

Coates: I take it the answer is none.

Louderback: I see my clients thinking very carefully about the services that they are going to be offering their customers, their depositors, thinking very carefully because if they offer the right kind of services and satisfy the right kinds of needs, they are going to be more successful than their competitors.

Coates: . . . if one says that competition will protect the consumer, I might be willing to accept that in the competitive EFT market, such as POS, point of sale, is today. But I don't think it's an adequate response in a more concentrated market, such as automatic clearing houses, where for preauthorized debits and credits there is only one game in town, through the association. [8, Chapter 19]

In addition to illustrating differences in perspectives on EFT research, this exchange illustrates the fact that the impacts of EFT on consumers might vary considerably depending upon the specific EFT technology being studied. For example, automated teller machines (ATM's) seem to have been well received by consumers whereas many point-of-sale (POS) terminals appear to have been poorly received by both consumers and retailers.<sup>1,15,39</sup> These differences in consumer reaction to different EFT technologies underscore the four key research issues in this area.

The first, as already suggested, is the *costs and benefits of EFT to the consumer*. Very little is known about how much EFT will cost, and how those costs will be allocated to consumers. It is clear that consumers eventually will bear most of the costs regardless of whether they pay through transaction charges or taxes for government subsidy of EFT development, or both. Similarly, little is known about what benefits EFT will bring to consumers, and how those benefits will be distributed among consumers. Much has been said in the promotional literature about the intended benefits of EFT on reduced costs of payment services, increased consumer convenience, increased security of financial transactions, and the like. However, the extent to which these

intended consequences actually occur is unclear; and the potential unintended and unanticipated consequences that might result are even less specified. And whether the purported benefits of EFT are desired by consumers is unknown.<sup>1,8</sup> More importantly, which group of consumers will receive the benefits and which will pay the costs? Humes<sup>20</sup> indicates that high-income, well-educated, financially sophisticated, credit-card-using consumers are the most likely users of EFT services. Yet, if costs are allocated generally over the consuming public and benefits accrue disproportionately to some minority of consumers, this might mean unfair "taxation" of those who do not benefit.

The second issue concerns the *consequences of EFT for the less advantaged*. It stems from the possibility that a large group of people might be excluded from EFT services. This group is comprised of disadvantaged, "unbanked" people who constitute 25 percent or more of the population.<sup>17,20</sup> EFT might either improve the access of these people to financial services, or it might have negative consequences for them, or both. On the positive side, for example, EFT might extend banking services through electronic means into areas not now adequately served by banks. On the negative side, EFT might provide new and abusive methods to garnish wages. It was argued at the conference that without specific government action the net impacts of EFT on the "unbanked" public are likely to be negative because the potential positive benefits will not accrue to them and because these people will not have the knowledge or means to change situations that are harmful to them. However, as Hiltz and Turoff indicate, this need not be the case:

"EFT represents an opportunity to purposely shape use of a new technology for social objectives as well as for corporate profits. EFT could facilitate basic changes in the nature and distribution of consumer financial services, and extend the benefits of such services to segments of the society which are currently at a disadvantage in dealing with existing financial institutions. At the very least, policy makers should take care that this new type of financial institution does not promote more inequality."<sup>17</sup>

More than any other, this issue illustrates the potential conflict between the values of the free enterprise perspective and more socially conscious neopopulist perspective. And probably more than any other issue, it was generally agreed that the *free enterprise model* was an insufficient means of insuring that the opportunity represented by EFT in this area was met.

The third issue requiring research concerns *the education of consumers for dealing with EFT*. Any effort to stimulate widespread adoption of EFT systems will require a coordinated educational effort of considerable scale.<sup>24,48,49</sup> It is critical that the educational effort enable consumers to make reasonable choices about whether to adopt the medium, and if so, how. Moreover, since EFT will introduce new levels of complexity in managing personal finances, it might be necessary to consider requiring education in the use of financial media in public schools much as driver's training is now required in many states. Research is needed, therefore, to determine the kinds of education necessary, to develop a knowledgeable public awareness of the EFT debates of the present, and assuming EFTs develop as anticipated, to facilitate knowledgeable and responsible use of the medium.

The fourth issue is *records control and consumer protection under EFT*. Control over financial records and prevention of fraud and other abuses are potential problems with EFT.<sup>24,38,41,42,46,47,48,54</sup> There is major concern today about the adequacy of consumer protection, as illustrated by a consumer representative's (Kathleen O'Reilly's) comment at the Conference:

"Consumers are becoming terribly concerned about the implications of the computer fraud phenomenon. It is far from science fiction. If there is not a commitment to the development of EFT systems that guarantee that appropriate (and available) technological methodology is used to minimize computer fraud (the prime victims of which are consumers), EFTs may well enhance the opportunity for that kind of dangerous abuse.

"What of the consumer concerns related to preauthorized payment? Despite the efficiencies that accompany preauthorized payments, there is still the concern that it reduces the actual spending pool availability of individual consumers. For some, such as the suddenly or temporarily unemployed, a real dilemma arises as to which bills to pay first."<sup>32</sup>

The changes introduced by EFT in regard to the philosophy and procedures of financial record-keeping and protection might be dramatic. One critical change is the definition of what constitutes "money" under an EFT system. If money is considered actual or symbolic, EFT represents a major move away from the use of cash to back up the symbols representing assets and liabilities. In such a case, what will be the standard for accounting for a given amount of money? An electronic impulse of certain characteristics over an authorized channel? This might have consequences for accounting, auditing and protection of financial records.

### *Economic impacts of EFT*

Two broad research areas were identified concerning the economic impact of EFT. The first dealt with the economics of EFT, specifically the issue of *the comparative costs of the current payments system and an EFT system*. The intensity of feelings about this issue is indicated by the fact that it was second highest in overall ranking by the Conference participants with a total of 268 points (Table III). However, much of the support for this issue came from the financial sector (44 percent, or 19 out of 43 people—see Table IV.) Eighty percent of the people (19 out of 24) at the Conference from the financial sector placed this issue in their top ten rankings. The other large group of participants at the Conference who ranked this issue among the top ten were from the federal government (47 percent or 8 out of 17 people). This difference in emphasis among the participants at the conference further illustrates the tension between private and consumer-related interests. For those directly involved in the implementation of EFT, questions concerning the direct costs of the payment system are of highest priority from both an institutional and a national perspective. For those more interested in the consumer impact, this issue is significant, but not nearly as important.

Al Lipis has noted that "Little information is available on the costs of cash transactions, yet such transactions constitute the bulk of payment transactions. Likewise, little is known about the costs of cash and checks to merchants."<sup>33</sup> The overall cost of the United States payment system in 1976 was estimated by Hamilton and Budd<sup>6</sup> to exceed \$22 billion, but we do not have a breakdown of the cost of specific areas of the system. The banking industry is therefore implementing EFT services without good comparisons of the overall costs and benefits relative to cash, checks and credit cards. In essence, there is little evidence today that EFT services are profitable or justified.

Information regarding the cost of our current payments systems is therefore needed to provide the basis for comparison of the costs and benefits of EFT and to provide a common ground of policy decisions. One problem is to determine the cost of cash transactions carried out. Another problem is to determine merchant costs related to the payment system. And still another problem is to discover the economics of banking services from the consumer's perspective.†

The second issue dealt with *the impact of EFT on the financial system and financial institutions*. This issue ranked twelfth with a total of 104 points (Table III). The primary question here is the impact of EFT on financial systems in

† There are different definitions in terms of what the costs are to the consumer for financial transactions. For example, Lipi<sup>33</sup> noted that Arther D. Little<sup>2</sup> studies estimated that the costs of handling checks to consumers were ten cents per transaction, whereas a study by Hamilton and Budd estimated that the costs per transaction from the consumer's perspective were 42 cents per transaction.<sup>6</sup> If more were known about the economics of consumer payment, we could better evaluate the costs and benefits of EFT from the consumer's perspective.

TABLE IV.—Ranking of Selected EFT Issues According to Institutional Perspectives

	PROVIDERS		REGULATORS		UNIVERSITY RESEARCHERS		SUPPLIERS		TOTAL
	Financial	Non-depository Institutions	State Government	Federal Government	Business/Economics Department	Other University	EFT Technology Industry	Other	
<u>Issue: Costs and Benefits of EFT to the Consumer</u>									
Number of people responding	24	1	2	17	2	9	6	11	72
Number of times ranked in the top ten	14	1	2	5	2	6	2	5	37
Total points	103	10	17	41	19	26	14	39	269
Intensity	7.4	10	8.5	8.2	9.5	4.3	7	7.8	7.
<u>Issue: Comparative Costs of the Current Payments System</u>									
Number of people responding	24	1	2	17	2	9	6	11	72
Number of times ranked in the top ten	19	1	2	8	2	3	4	4	43
Total points	119	8	12	42	19	14	25	29	268
Intensity	6.2	8	6	5.3	9.5	4.7	6.3	7.3	6.
<u>Issue: Definition of EFT Regulation: What Should Be Regulated?</u>									
Number of people responding	24	1	2	17	2	9	6	11	72
Number of times ranked in the top ten	11	0	1	10	1	4	3	3	33
Total points	64	0	3	79	7	4	24	21	202
Intensity	5.8	0	3	7.9	7	1	8	7	6.
<u>Issue: EFT Impact on the Long-Range Character of Society</u>									
Number of people responding	24	1	2	17	2	9	16	11	72
Number of times ranked in the top ten	9	0	2	4	0	7	2	6	30
Total points	40	0	15	39	0	54	9	42	199
Intensity	4.4	0	7.5	9.8	0	7.7	4.5	7	6.
<u>Issue: EFT as a Case Study of Technological Change and Impact</u>									
Number of people responding	24	1	2	17	2	9	6	11	72
Number of times ranked in the top ten	7	1	0	7	2	7	3	4	31
Total points	14	6	0	11	9	46	13	17	116
Intensity	2	6	0	1.6	4.5	6.6	4.3	4.3	3.

general, and competing financial institutions in particular. Our current financial system is already undergoing a variety of changes and pressures unrelated to EFT. However, EFT complicates the process and provides added pressure for change:

"EFT is clearly an important part of the change process which is under way. In fact, unlike regulatory or legislative innovations which must evolve slowly through the jockeying of various special interest groups, EFT offers individual financial institutions the opportunity to try to gain a larger share of the retail-banking market. For example, since S & L's are less limited in terms of branching, with EFT they could leapfrog the paper-check system by providing bill-payer services and developing, or linking into, a system of off-premise terminals. Although the evolution of EFT is undoubtedly moving at a slower pace than some expected, the movement is under way as a part of a larger evolution. If EFT were the only change that was occurring, then it might be possible for progress on that front to stop. However, EFT is both a fuel and a passenger in the movement of innovation, and the whole of the change process is greater than the sum of the parts."<sup>9</sup>

What will be the impact of EFT on the comparative market share for different types of financial institutions? To the extent that EFT requires economies of scale, what impact will changes in technology have on smaller institutions? And finally, what impact will EFT have on the overall structure of financial institutions?<sup>††</sup>

#### *Regulation and control of EFT*

Our current system of financial regulation and control is a complex and controversial topic. The United States has a unique system of private financial intermediaries and markets. Compared to other nations, we have a wide range of financial institutions with 14,000 commercial banks, 5,000 savings and loan associations, 500 mutual savings banks and 22,000 credit unions. The dual banking system is one of the cherished aspects of our financial structure, and, over the years, our government has tended to define and supervise the activities of financial institutions in an overlapping and sometimes competing fashion.<sup>9</sup>

The importance of regulatory issues is suggested by the fact that the issue of "defining EFT regulation and what should be regulated" was ranked third highest by the Conference participants, receiving a total of 202 points, and in terms of overall intensity it was the fifth highest. (Table III.) However, ranking of importance seemed to vary. The issue received particularly strong support from the representatives at the Conference from the federal government. Of the 33

<sup>††</sup> As a final note, few people highlighted the impact of EFT on money and monetary flows as an issue which required priority research attention. There was concern about how we define money and about the influences of EFT on overall monetary policy. However, most of these questions are not questions concerning EFT *per se*, rather they are questions having to do with basic monetary policy, of which EFT is only a part and, therefore, probably need to be considered part of broader research questions dealing with monetary policy. [For a further discussion of this, see 8, 14.]

people who ranked this issue in the top 10, 10 people were from the federal government (Table IV). Like the issue on comparative costs, EFT regulation was ranked highest when viewed purely from an institutional perspective. When viewed from a national perspective it was only of medium importance.

Once again, then, views differed concerning the amount of regulation appropriate for EFT depending upon the value perspective of the participants. [Also see 24,29,49.] One view would rely on the *free enterprise model* to evaluate whether or not EFT innovations were effective by determining whether or not consumers were willing to accept the changes. Those arguing from this perspective find that regulations often retard development, and that too much research might have a negative impact. The other perspective, essentially a *statist model* of government responsibility, argued that the concerns about potential problems could be so great that regulation might be used to slow down development and to avoid "nonexistent" problems. The following dialogue illustrates this debate:

Benton: is there any reason for the government as an actor to be participating in such a way so that electronic payment systems come into being? The government is supposed to get involved in matters that are of national importance. Is there any evidence that this EFT is that significant?

Cox: I think the fact that we are dealing with something that is closely related to one of the social instruments for control . . . of the health of our economy is the reason why the government has reason to be more interested in the payment system than in some other aspects of our society.

Coates: Government should interfere when there are significant externalities . . . Insofar as electronic funds transfer have effects which are outside the chain of buyers and sellers, and insofar as they become large, one has the argument for government intervention because that is the only alternative mechanism society has for dealing with externalities.

Reistad: The easiest way to determine which of those externalities should be grappled with would be to go into the pioneering of the system and see what evolves from it.<sup>7</sup>

Thus, a range of special and competing perspectives emerge, and many of the issues concerning the first issue in this area—regulation and what should be regulated—are ultimately political judgments. While final policy decisions will be made in the arena of politics and policy choice, research may help to frame the debate and to reveal the consequences of alternative choices.

The second regulatory issue concerns *the appropriate mode for regulation and the need for restructuring regulatory institutions*. It is frequently stated by the promoters of EFT and the *private enterprise model* that "the marketplace can serve as an effective regulator of EFT." Yet, there is little evidence either for or against this assertion. Moreover, as noted above, we have an overlapping system of state and federal regulations with notable inefficiencies and problems. Consequently, serious questions exist regarding what should be the relationship between state and federal regulators in the future, and what range of options is available for organ-

izing the regulatory structure related to financial institutions and EFT.

Assuming that the decision is made that regulation is required beyond the marketplace, it might be possible to study the range and options of organizing EFT-related regulatory structures. Without institutional restructuring, future regulations are likely to look similar to the regulations of the present and the past. However, perhaps regulatory reform within the financial system is appropriate and a reworking of the existing regulatory strategy may be necessary. Decisions about such reworking could be aided by knowledge about possible reforms and about the value of taking a reform approach as opposed to an amendment approach to existing regulatory policy.

#### *Monitoring and evaluating EFT systems in the broader context*

One of the critical areas for future EFT research relates to the long-term and broad social impacts of EFT, and at the Conference two such issues received particularly strong attention: The first, "EFT and Social Change" (dealing with the potential for EFT to bring considerable long range change in the social and institutional character of society), rated fourth in terms of total points; and the second, "EFT as a Case Study of Technological Change and Impact," was tenth (Table III).

Although the two issues were relatively close in terms of number of times ranked in the top ten and total points, "EFT and social change" received a much higher intensity rating. In fact, its rating of 6.6 was the second highest intensity rating in the entire survey (Table III). This seems reasonable when realizing that this issue deals with the broad question of EFT and social change, and "EFT as a case study of technology change" deals with a specific research approach. Intensity concerning the broad issue is high, whereas the specific case study approach was felt to be important by many, but not as intensely.

Of the people who voted for "EFT and social change," support was particularly strong from the university and financial communities with comparatively little interest from the federal government (although those from the federal government who were interested gave it a high intensity rating) (Table IV). Of those who voted for "EFT as a case study" support was again strong from the university community with 9 out of 11 people ranking it in the top ten (Table IV).

The issue of "EFT and Social Change" concerns the long-run interactions between society and EFT. Most discussions of EFT focus on likely impacts that EFT will have on society, assuming technology to be largely a deterministic and *free-market* force in the society. However, the *statist perspective* argues that society also will affect the technology—its regulation, development, use and impact. This difference in perspective is important because the former view assumes technological determinism, whereas the latter view assumes that EFT technology is an instrument of society and that its impact will be importantly shaped by its interaction with long-term social trends. Coates' criticism of current EFT research is most revealing in this regard:

Coates: The most important limitations I see on the work to date is the absence of any image of the future. There is little awareness one can sense in the reports of the NCEFT that the nation is in a state of major evolution. There is no awareness of the many long-term trends which are remaking our society. There is no sensitivity to the fact that EFT is part of those trends both as an influence and as an effect. This absence of a vision or framework of the future is the single most critical deficiency in the Commission's work.

The second structural deficiency in the work is a near total absence of any general principles which could form a conceptual and analytical framework. For example, a general principle that would have been most useful is that the primary political, civil libertarian and constitutional risks for the American people in the next three decades are from government itself. With that principle informing the deliberations of the Commission, many of its conclusions, I believe, would have come out differently. There would have been a series of sharp and useful distinctions made between the privacy violations or potentials for such violations from the private sector and the much more fundamental and serious violations from government. This incidentally ties in with one's view of the future.<sup>7</sup>

In order to understand the impacts of EFT in society, it is important to identify the broad social trends emerging in the future which will interact with this new technology. [8, Chapter 14; 28] Three inter-related questions are involved. The first concerns what long-run changes in society are expected to occur that will interact with EFT. These include changes in life styles, housing preferences, work patterns, transportation and communication systems, retailing and shopping patterns, and similar social patterns. The second question concerns how these changes will interact with EFT systems, and with one another, to affect how EFT systems will be used. The third question concerns what impacts EFT might be expected to have in the society given different scenarios and patterns of interaction. For example, what effect will EFT, possibly in conjunction with other transportation and communication networks, have on social mobility of individuals and households at different strata of society? On work patterns? On shopping? On life in the home?

The second research issue focuses on *EFT as a case study of technological change and impact*. Whereas the foregoing research issue concerns forecasting likely interactions and impacts of EFT, this issue concerns baseline measurement and longitudinal monitoring of the actual impacts of EFT and the study of EFT as a general illustration of the interaction of technology, society and public policy. EFT might be established as a case for study on a continuing basis, in much the same way that weather and public political opinions are now studied. Specifically, EFT affords an opportunity to begin serious research into several general questions about technology and society: how technology emerges; how new technology is handled by existing institutions; how new technologies are assimilated by these institutions and by the public; what specific impacts new technologies have; how new technologies create synergistic impacts with other technologies; and how technologies change over time to conform to new circumstances and developments. Each of these represents an aspect of EFT concern that has academic,



policy and practical relevance if carried out over time. [For a further discussion of this topic, see 8, Chapter 14.]

## CONCLUSION

Only a few years ago, many technologists and financial experts predicted that electronic funds transfer systems would usher in the checkless-cashless society. But it hasn't happened. The early predictions about the impact of EFT on the future of the financial system have gone far wide of the mark.<sup>1,20,41</sup> Instead of electronic banking, we have a greatly improved paper-based financial system—one that relies on cash and checks even more heavily than in the past. Thus, EFT technologies are following the same model of innovation diffusion that earlier characterized the introduction and spread of general purpose computer technologies. The introduction of computers in the 1950's was followed by enthusiastic predictions about how office work and managerial work would be revolutionized. Yet, as we approach the 1980's, research has made it clear that many of the positive impacts of computers in the office and the board room have yet to be realized, if they ever will be, and many of the dire impacts have not occurred. The potential for major change may still be there, but the incorporation and routinization of the technology is occurring much more slowly than the early technologists and promoters expected. And most importantly, the technology is being shaped by the organizational and institutional context in which it is used rather than solely serving as a driving force of its own which dramatically reshapes its context. The technology has come to be recognized as simply a tool which is shaped more by the agendas of those who would use it than by the possibilities inherent in the technology. [For example, see 78,83.] EFT systems, therefore, may be expected to exhibit a similar evolution through many small incremental changes and adaptations of the technology to changing organizational and societal definitions of its appropriate use.

From the standpoint of public policy, this means that issues surrounding the introduction and use of EFT will remain on the public decision agenda for years to come. The National Commission on EFT is over, but many of the policy issues which it raised have not been settled. Even for those issues which seemingly have been settled, it is likely that new understandings and new experiences with the effects of current public policy will generate recognition of the need for new policy in the future. And policy and technology will be mixed. As EFT technology itself changes and is adapted to new uses, public policy will be needed to deal with the impacts of the technology. But as we develop greater understanding of the ways in which the society might utilize the technology, public policy also will be directed increasingly toward shaping the way the technology is used.

From the standpoint of research, the joint evolution of the technology and the public agenda means that continuous monitoring and evaluation of EFT is paramount. Objective, scientific information and knowledge can go a long way toward informing the public decision agenda over the next several years and even decades of policy-technology evolution.

In looking back over the agenda, it is important to realize that most of the recommended research is in a middle-time range, probably over the next five to ten years. Only the last two recommendations call for long-term research and for the institutionalization of research regarding the evolution of EFT in the United States. Middle-range research clearly is required and significant. It is important to begin systematic research on EFT systems to continue the momentum created by the National Commission on EFT and to fill the many gaps in needed information faced by the Commission. There is also a need to create a cadre of people who are knowledgeable about EFT systems and are capable of doing objective, scientific research in the area, rather than solely client-oriented studies. Much of this research should begin with smaller, specific questions which can be successfully answered within a moderate time frame both as a means of building knowledge and as a means of informing policy in the near future.

However, it is also important to remember that this paper and the research agenda setting process have clearly demonstrated that values toward the evolution of technology and research concerning this evolution vary significantly depending on individual and institutional perspectives. Those interested in the development of the technology will establish their own momentum and information base to justify their movement. It is therefore essential that more than short or even middle-range research receive attention. Long-term research is essential along with the continual monitoring and evaluating of the evolution of EFT.

## REFERENCES

1. "A retreat from the cashless society," *Business Week*, April 18, 1977, pp. 80-90.
2. Arthur D. Little, Inc., *The Consequences of Electronic Funds Transfer: A Technology Assessment of Movement Toward a Less Cash/Less Check*, Rep. C-76397, Cambridge, Mass., January 1975.
3. Balderston, F. E., Carman, J. and Hoggatt, A. "Computers in Banking and Marketing," *Science* 195, March 18, 1977, pp. 1115-1118.
4. Benton, J., "Electronic funds transfer: pitfalls and payoffs," *Harvard Business Rev.* 55, July-August 1977, pp. 16-19, 28-29, 164-173.
5. Boucher, W. I., *A Comment on EFT Research: Five Tries and a Start*, Working Paper No. PPR-7704, Public Systems Evaluation, Inc., Cambridge, Mass., 1976.
6. Budd, G. and Hamilton, E., "The Economics of the Payments Mechanism," unpublished article, 1976.
7. Coates, J. F., Cox, E. B., and Reistad, D. L., "Monitoring EFT in Evolution," in Colton, K. and Kraemer, F., *Computers and Banking*, Plenum Publishing Corporation, 1980.
8. Colton, K. and Kraemar, K., *Computers and Banking: Electronic Funds Transfer Systems and Public Policy*, Plenum Publishing Corporation, New York, 1980.
9. Colton, K., "EFT and the Process of Change," in Colton, K. and Kraemer, K., *Computers and Banking*, Plenum Publishing Corporation, 1980.
10. Cox, E. B. and Giese, P., "Now it's the less-check society," *Harvard Business Rev.* 50, November-December 1972, pp. 6-18.
11. Ege, S. M., "Electronics funds transfer: A survey of problems and prospects in 1975," *Maryland Law Rev.* 35, 1, 1975, pp. 3-56.
12. Eisenbeis, R. A. and Wolkowitz, B., "Sharing and Access Issues," *The Banker's Magazine*, 162, 2, March-April 1979.
13. Ellul, J., *The Technological Society*, Knopf, New York, 1974.
14. Federal Reserve Bank of Boston, "The economics of a national electron funds transfer system," *Conf. Ser. No. 13*, Boston, Mass., October 1974.

15. Fisher, J. F., "EFT—The Decade of the 1980's: New Concepts for the World of Banking," *The Banker's Magazine*, 162, 2, March-April 1979, pp. 21-24.
16. Freeman, D., *Technology and Society: Issues in Assessment, Conflict and Choice*, Rand McNally, Chicago, Ill., 1974.
17. Hiltz, S. R. and Turoff, M., "EFT and social stratification in the USA: More inequality?" *Telecommunications Policy* 2, March 1978, pp. 22-31.
18. Hoffman, L., *Modern Methods for Computer Security and Privacy*, Prentice-Hall, Englewood-Cliffs, N.J., 1977.
19. Horan, T. F., "Outlook for EFT Technology," in Colton, K. and Kraemer, K., *Computers and Banking*, Plenum Publishing Corporation, 1980.
20. Humes, K. H., "The Checkless/Cashless Society? Don't Bank on It!" *The Futurist*, October 1978, pp. 301-306.
21. Johnson, H. G. and Arnold, E. C., "Preparing for EFT," *The Bankers Magazine*, 162, 2, March-April 1979, pp. 43-45.
22. Kaufman, G., *Money, the Financial System, and the Economy*, Rand McNally, Chicago, Ill., 1973.
23. King, J. and Kraemer, K., "Electronic funds transfer as a subject of study in technology, society and public policy," *Telecommunications Policy* 2, 1, March 1978, pp. 13-21.
24. Kling, R., "Passing the digital buck: Unresolved social and technical issues in electronics funds transfer systems," TR #87, Inform. and Comptr. Sci. Dept., U. of California, Irvine, Calif., June 1976.
25. Kling, R., "EFTs: social and technical issues," *Computers and Society* 7, 3, Fall 1976, pp. 3-10.
26. Kling, R., "Six models for the social accountability of computing," *Computers and Society* 9, 1, Summer 1978.
27. Kling, R., "Information systems and policy making: Computer technology and organizational arrangements," *Telecommunications Policy* 2, 6, March 1978, pp. 22-32.
28. Kling, R., "EFT and the quality of life," *Proc. AFIPS 1978 NCC*, Vol. 47, AFIPS Press, Montvale, N.J., pp. 191-197.
29. Kling, R., "Value Conflicts and Social Choice in Electronics Funds Transfer System Developments," *Communications of the ACM* 21, 8, August 1978, pp. 642-657.
30. Kohn, S. J., "Managing the EFT Decision," *The Banker's Magazine*, 162, 2, March-April 1979, pp. 25-32.
31. Kraemer, K. L., King, J. L., and Colton, K. C., "Towards an Agenda for EFT Research," *Computers and Society* 8, 2, Summer 1977, pp. 3-11.
32. Leary, F. Jr., O'Reilly, K., and Palmer, H. M., "Discussions of papers," by K. Humes, S. R. Hiltz and M. Turoff, Working Paper No. PPR-7709, Public Systems Evaluation, Inc., Cambridge, Mass., 1976.
33. Lipis, A., "Costs of the Current U.S. Payment System," in Colton, K. and Kraemer, K., *Computers and Banking*, Plenum Publishing Corp., 1980.
34. Louderback, P. D., "Status of EFT: An Assessment of Services and a Review of EFT in the Fifty States," in Colton, K. and Kraemer, K., *Computers and Banking*, Plenum Publishing Corporation, 1980.
35. National Commission on Electronic Fund Transfers, "EFT and the Public Interest," U.S. Gov't. Printing Office, Washington, D.C., February 1977.
36. National Commission on Electronic Fund Transfers, "EFT in the United States: Policy Recommendations and the Public Interest," U.S. Gov't. Printing Office, Washington, D.C., October 1977.
37. Nilson, H. D., "The Future of Credit Cards," *The Bankers Magazine*, 162, 2, March-April 1979, pp. 54-60.
38. Parker, D. B., *Crime by Computer*, Scribners, New York, 1976.
39. Pastore, S., "EFT and the Consumer," *The Bankers Magazine* 162, 2, March-April 1979, pp. 35-42.
40. Peat, Marwick, Mitchell and Co., *EFT: A Strategy Perspective*, Peat, Marwick, Mitchell and Co., New York, 1977.
41. Portway, P. S., "EFT systems? No thanks, not yet," *Computerworld* 12, 2, January 9, 1978, pp. 14-16, 21, 23-25.
42. Privacy Protection Study Commission, "Personal Privacy in an Information Society," U.S. Gov't. Printing Office, Washington, D.C., July 1977.
43. Prives, D., "The Explosion of State Laws on Electronic Fund Transfer Systems," P-76-1, Prog. Inform. Technologies and Public Policy, Harvard U., Cambridge, Mass., 1976.
44. Richardson, D. W., *Electric Money: Evolution of an Electronic Funds Transfer System*, M.I.T. Press, Cambridge, Mass., 1970.
45. Rose, S., "More Bang for the Buck: The Magic of Electronic Banking," *Fortune* 95, 5, 1977, pp. 202-226.
46. Rossman, L. W., "Financial industry sees EFT privacy laws adequate," *American Banker* CXXI, 210, October 28, 1976, pp. 1, 11.
47. Rule, J., *Private Lives and Public Surveillance*, Schocken Books, New York, 1974.
48. Rule, J., "Value Choices in Electronic Funds Transfer Policy," Office of Telecommunications Policy, Executive Office of the President, Washington, D.C., October 1975.
49. Sayre, K. (ed.), *Values in the Electric Power Industry*, U. of Notre Dame Press, Notre Dame, Ind., 1977.
50. "The time is NOW," *Forbes Magazine* 120, 1, July 1, 1977, pp. 61-62.
51. Turoff, M. and Mitroff, I., "A case study of technology assessment applied to the cashless society concept," *Technol. Forecasting Soc., Change* 7, 1975, pp. 317-325.
52. U.S. Dept. HEW, Secretary's Advisory Committee on Automated Personal Data Systems, Records Computers, and the Rights of Citizens, Washington, D.C., 1973.
53. Weizenbaum, J., *Computer Power and Human Reason*, Freeman, San Francisco, 1976.
54. Westin, A. and Baker, M., *Databanks in a Free Society*, Quadrangle Books, New York, 1972.
55. Winner, L., *Autonomous Technology: Technology Out-of-Control as a Theme in Political Thought*, M.I.T. Press, Cambridge, Mass., 1977.



# A linear programming model for optimal computer network protocol design\*

by JOHN F. HEAFNER and FRANCES H. NIELSEN

National Bureau of Standards  
Washington, D.C.

## INTRODUCTION

After providing background information on protocol modeling, we then describe a linear programming model useful in protocol design. This is followed by a description of its use and some comments on its implementation.

### *Layered protocols*

The ISO reference model<sup>15</sup> defines seven layers of protocols. The lower three layers address communications within the backbone network(s). Layers four through seven mainly service the host computer environments, both from the standpoint of communications and data processing. The linear programming model described here pertains to the protocols found in layers 4-7. As prescribed by the reference model, layer 4 (transport control) provides reliable transfer of information between host environments. Layer 5 (session control) supports the dialogue of higher level protocols and processes. Layer 6 (presentation control) provides data transformations from code translation through record restructuring of the information being transferred. Layer 7 (application control) contains application-oriented protocols and is the layer with which most application processes directly interface.

The reference model assigns precise meaning to the terms protocol and interface. We speak of the *protocol* as peer layer communication. For example, protocol dialogue takes place between an entity at the session layer in one system and the corresponding entity at the session layer in another system. Communication between adjacent layers, e.g., between one entity in session control in one system and another entity in transport control in the same system, is known as an *interface*. Given these terms, we characterize a protocol *service feature* as a singularly identifiable capability or service provided by a protocol to higher level protocols or pro-

cesses. Service features are "visible" across the upper adjacent interface. Our interest lies in designing protocols by first specifying their service features.

### *Protocol families and application categories*

A number of different protocols are needed at each layer. For example, within the application layer we find (portions of) Remote Job Entry (RJE), the Common Command Language (CCL) of a File Transfer Protocol (FTP), voice transmission protocol, and perhaps parts of a Network Virtual Terminal protocol, to name a few. More germane to the linear programming model, we anticipate the need for variants of, for example, the FTP (i.e., a *family* of FTPs) to satisfy the requirements of diverse application categories found throughout industry and within government agencies.

To further the notion of a family of protocols, consider the *kernel* FTP, which we loosely define as the protocol providing only the service features (i.e., the service operations as seen by the adjacent entities above the FTP) essential to most of the using processes most of the time. The kernel protocol is the "barebones" service needed to conduct many higher level missions. Now, imagine enhancing this kernel FTP by providing additional service capabilities. By varying the added capabilities to suit individual applications a family of FTPs can be derived. Assuming the kernel capability exists, we will describe an analytical tool to aid in specifying an extended, target protocol belonging to some specific family.

### *The problem of designing a protocol family*

Optimal design of a protocol for an application using a linear programming model is a subtask of the larger problem of determining a minimal family of protocols for a whole range of application classes. This larger problem of deciding an entire protocol family rests on the following postulates.

1. All the categories of applications can be identified.
2. Service features in the kernel of some protocol of interest can be determined.

\* This work is a contribution of the National Bureau of Standards and is not subject to copyright. 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 for the purpose.

3. All features beyond the kernel that could be provided within the family can be identified.
4. For each application category the necessary and desirable features can be listed.

Figure 1 captures these assumptions. Given these assumptions, the problem remains to select the minimal set of protocols within the family that will satisfy all applications' requirements. This is illustrated by Figure 2. There are two opposing constraints. On one hand, it is not economical for an application to carry the overhead of using a protocol containing features of no intrinsic value to the application. Conversely, we want as few protocol standards within the family as possible, consistent with the first constraint. It is necessary that features  $(f_{i1}, \dots, f_{ik})$  essential to application category  $A_i$  be in some protocol  $P_q$  if  $A_i$  is satisfied by that  $P_q$ . Note, however, that some  $f_{ij}$  can appear in both  $P_q$  and some other  $P_r$ .

Thus, choosing the family of standard protocols, e.g., for FTP, is a higher level problem, one which can be viewed as an optimization problem in operations research. We identify the higher level problem in order to substantiate the need to solve the subtask implied by assumption four, which this paper addresses.

*Solution methodologies for some subproblems*

To deliberate the feeder problem of assumption four above, we must convince ourselves that the first three assumptions, upon which it depends, can be satisfied.

The first assumption was that application categories could be named. Today, we have no such list. However, these data may be obtained by applying survey research methods to the appropriate communities. Also, the International Data Corporation (IDC) has identified 100 applications of data processing in industry.<sup>13</sup> Such a list might serve as a base from which to distill government and industry application groupings.

Assumptions two and three state that we can identify the family kernel and extended services. Enumerating service features is presently an unsolved problem. At the outset, experts do not agree on exactly what service features are. To compound the problem of exhaustively listing features we wish to entertain both in place and proposed protocols. The document common to the two is a protocol specification, as opposed to a protocol design document needed for implementation. Recently, work has been done toward defining rules for extracting features from protocol specifications.<sup>11</sup> Thus, we tentatively accept that a family kernel and exten-

APPLICATION CLASSES	ESSENTIAL FEATURES OUTSIDE KERNEL	DESIRABLE FEATURES OUTSIDE KERNEL
$A_1$	$\{f_{11}, f_{12}, \dots, f_{1k}\}$	$\{f_{1(k+1)}, f_{1(k+2)}, \dots, f_{1n}\}$
$A_2$	$\{f_{21}, f_{22}, \dots, f_{2k}\}$	$\{f_{2(k+1)}, f_{2(k+2)}, \dots, f_{2n}\}$
•	•	•
•	•	•
•	•	•
$A_m$	$\{f_{m1}, f_{m2}, \dots, f_{mj}\}$	$\{f_{m(j+1)}, f_{m(j+2)}, \dots, f_{mp}\}$

**NOTE: THE KERNEL FEATURES ARE COMMON TO ALL  $A_j$ .**

Figure 1—Association of protocol features with application classes.

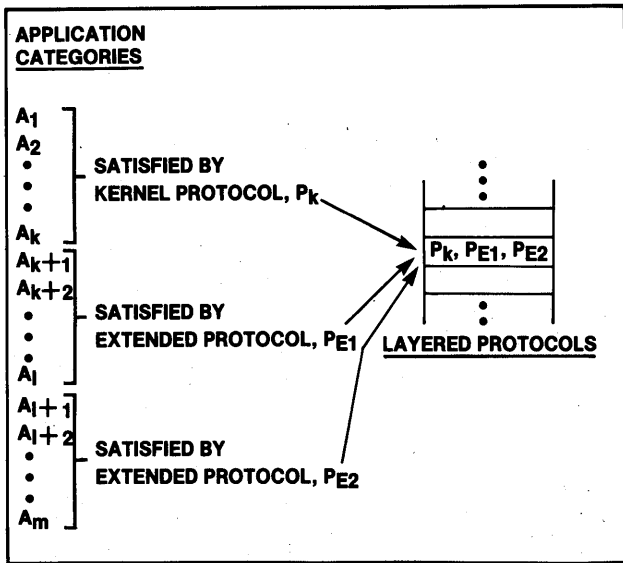


Figure 2—A protocol family satisfying application requirements.

sions can be identified by: 1) extracting features from many documents, 2) working with application builders and users to determine essential and desirable features for each application category, *without regard to costs*, and 3) then taking perhaps the intersection of features of the examined protocols as the kernel.

THE LP MODEL FOR PROTOCOL DESIGN

Our fourth assumption was that necessary and useful features of each application category were obtainable. Our work considers this subproblem—associating features with applications. In particular, the aim is to derive the information contained in Figure 1 on an application class basis so that subsequent methods can develop the families indicated in Figure 2. In recent years, advances have been made in designing application and support software by including the prospective user in the design stages.<sup>1,5,9,10</sup> These efforts have concentrated on needed and desired services. Yet, inputs from prospective users have not been conditioned by the costs of providing these services. *The LP approach directly involves the user in the design loop, and importantly, it factors in costs of providing services as well as value or benefit derived therefrom.*

A canonical linear programming model

Figure 3 portrays one standard form of a linear programming model.\*\* The rows of the matrix represent resources; the columns depict activities or events that consume the resources. Hence, the matrix elements,  $a_{ij}$ , are unit resource

\*\* Although it is not clear that linear relationships hold among features, the LP model provides a useful starting point from which to investigate protocol quantification methods.

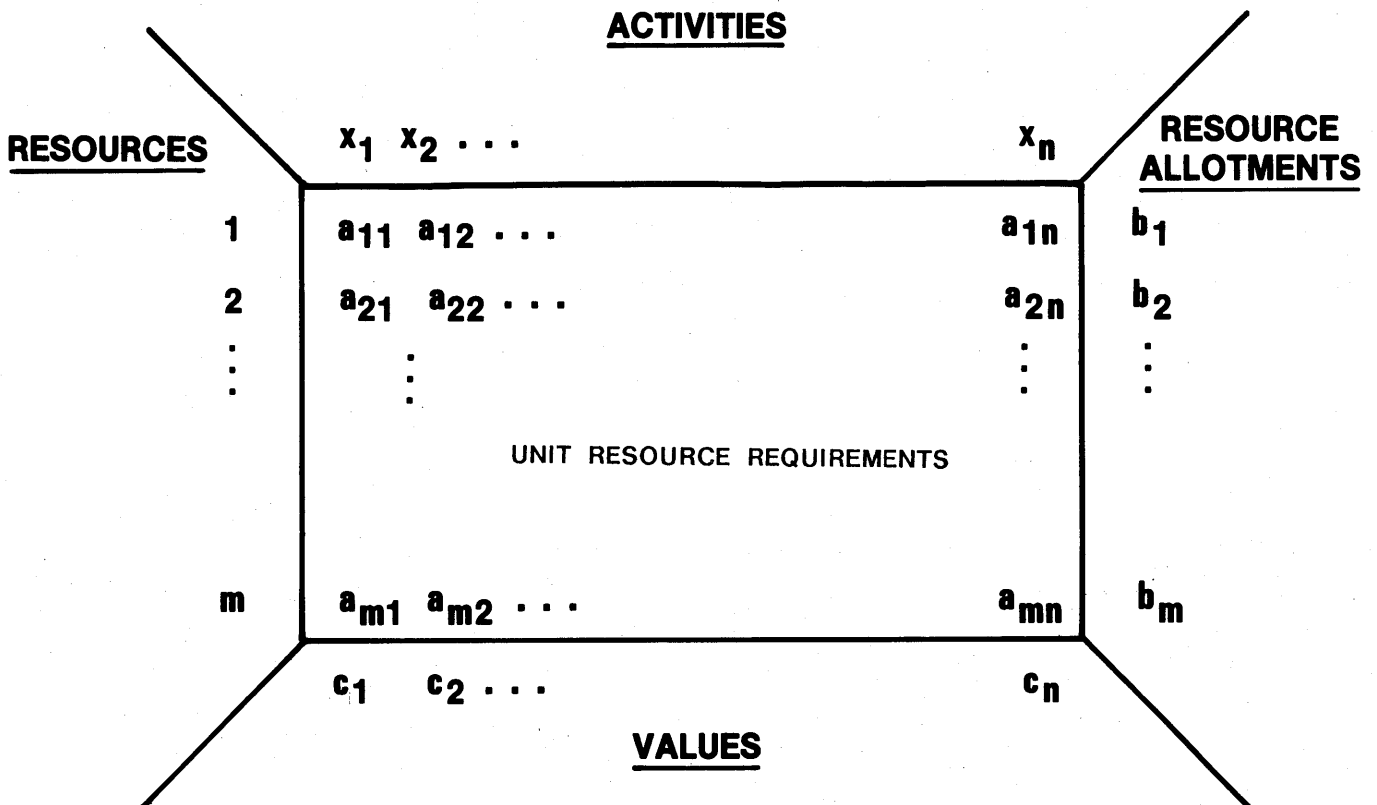


Figure 3—A standard linear programming model.

requirements, i.e., the amount of resource  $i$  consumed by one unit of activity  $j$ . Elements of the row vector,  $C$ , represent the benefit or value of the units of the corresponding activity ( $x$ ). Prior to using the model, the permissible resources and activities are defined and costs and values are established. To use the model for assigning resources to activities, the column vector  $B$ —the amount of each resource to be made available—is specified. The model then maximizes the return, called the objective function:

$$Z = \sum_{j=1}^n c_j x_j$$

subject to the constraint

$$\sum a_{ij} x_j < b_i, \quad 1 < i < m.$$

*The LP model for protocol design*

The model shown in Figure 3 is readily adapted to relate costs to values in the specification of a protocol. Figure 4 illustrates the revised representation. Resources are present as in the standard form. Notice that they are partitioned into meaningful groups to allow collective constraint statements to be made. For example, total development costs can be

stated rather than or in addition to restricting individual developmental costs. In the other dimension of the matrix, activities are replaced by nonkernel protocol service features of a given family, such as the FTP family.

Thus, Figure 4 represents only that part of the data base appropriate to FTPs. (If designing other protocols such as a Network Virtual Terminal or a Host-Host Protocol, envision a third dimension of the matrix whose elements correspond to the feature/resource costs of those protocol families' features.) The protocol layer and family are input parameters to the model, used to select the appropriate subset of the data base for use. Note also that the data base contains only improvement features: we presuppose that the kernel has been established. The model is used only as an aid in defining the extended capability protocols (for application categories as shown in Figure 1). The value vector  $C$  shown in Figure 4 represents the values associated with a given application group and protocol layer. Thus, for an instance of use of the model the vector is selected from the larger data base representing values for each application category/protocol layer. The dependency of value on both parameters, protocol layer and application category, is easily demonstrated. Consider the applications of banking and network voice conferencing. Clearly, data compression is a feature that has a higher value to the conferencing application

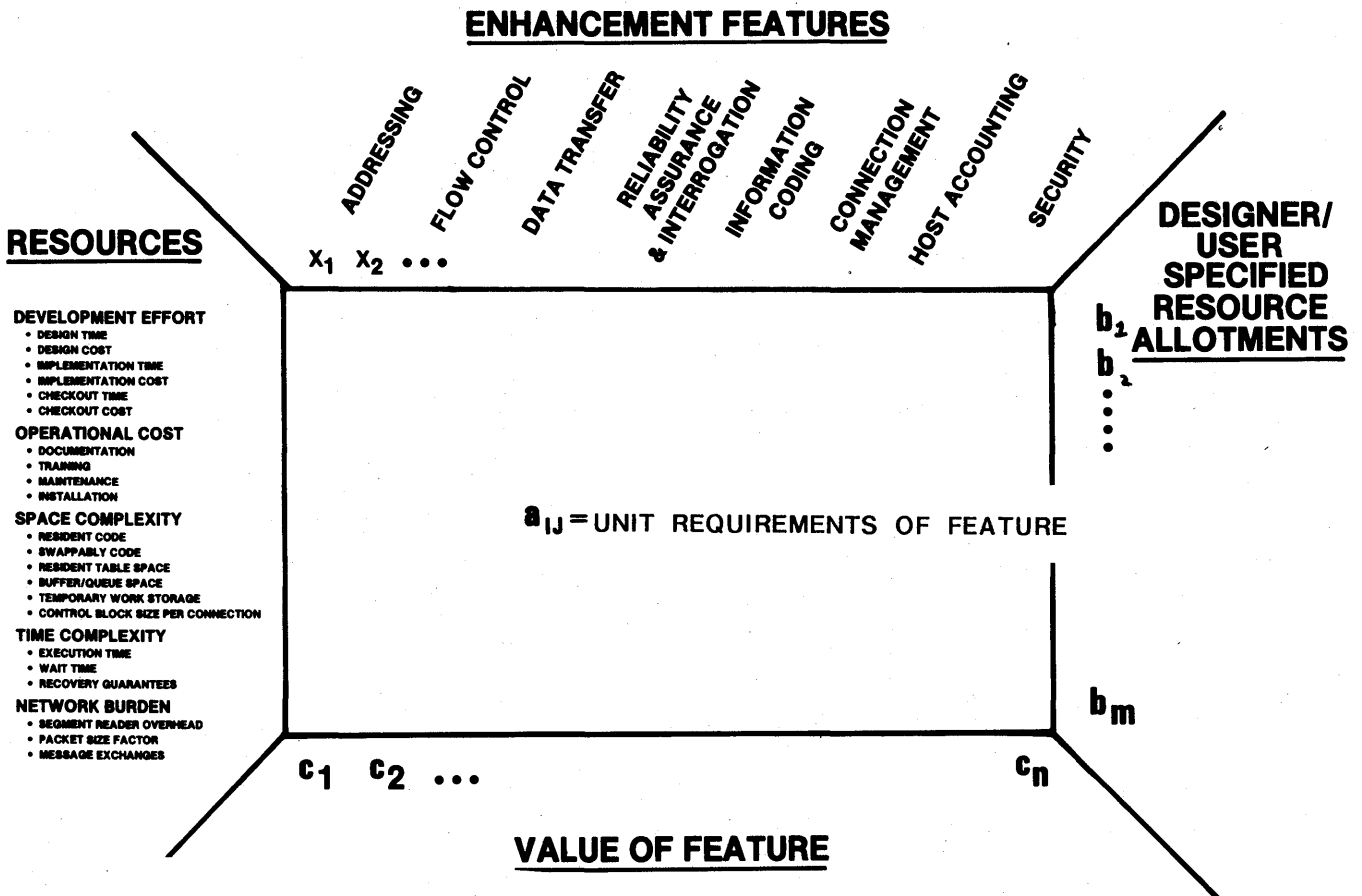


Figure 4—Linear programming model for protocol development.

than to the transaction processing application. The dependency upon protocol layer is equally valid. When designing an FTP protocol (at layer 6 or 7) the application builder certainly desires reliable transmission between remote and local hosts. However, end-to-end error control is generally associated with the transport control (layer 4). Thus, if designing higher level protocols, this feature's value (if the feature appears in this portion of the data base) would be multiplied by a layer coefficient near 0, since error control is assumed to be managed in the kernel of the transport protocol(s).

#### *Determination of costs and values*

Before the model may be used as an aid in protocol design, the  $a_{ij}$  and  $c_j$  of Figure 4 must be ascertained. To obtain them, we assume that a reasonable list of resource types can be obtained from protocol designers. Those of Figure 4 are indicative, not exhaustive. Enhancement features can be obtained by applying extraction rules<sup>11</sup> to a reasonably large number of protocol specification documents and omitting kernel features. Now, the costs,  $a_{ij}$ , can be supplied by protocol designers either from direct implementation and operation experience or through some other technique. An approximation to values (independent of costs) can be obtained from application users and builders via survey research methods.

We can now use the model to design a (mathematically) optimal protocol within a family and for a particular application.

#### USE OF THE MODEL

##### *First, a "toy" protocol design*

Use of the model is shown by the following hypothetical example. We wish to extend the kernel of the File Transfer Protocol (FTP) for an application that transfers extremely large files, such as those typical of some NASA applications. To enhance the FTP kernel in support of transferring very large files, the application builder states for the protocol designer: 1) the kinds of extensions he needs, 2) their worth to him, and 3) permissible costs. The protocol designer then translates these inputs into parameters for the LP model which then generates a protocol design. The protocol is optimized in that the objective function is maximized for the input parameters. The LP output, namely a list of features to incorporate in the extended protocol, is an approximation. The process is then iterated until both parties are satisfied with the resulting feature list.

The following scenario is typical of one iterative use of the model. Application builder specifications:

1. I want the file transfer program improved only in terms of the data transfer phase.
2. In particular, I want more detailed and meaningful responses to detected errors and to status inquiries.

3. I am getting real time satellite data coming into a remote host computer. It is then netted via the FTP to my local host. Some of the records being transferred are much more critical than others. That is, the redundancy and granularity of the data differ over record types. Thus, I need the ability to dynamically change the unit sizes of rollback or recovery of information upon network failure, and I need the ability to dynamically change the size of the commitment unit, i.e., the amount of information guaranteed to be transmitted within the session.
4. I do not especially need any more default options on the new data transfer service.
5. I will never change the size of the quarantine unit, that is, the size of what I refer to as records, since my records are all the same fixed size.
6. I am willing to spend a total of \$30K on developing these improvements.
7. I am willing to forgo about 5 percent in operational costs for these improvements.

The protocol designer now translates these requirements stated in English into mathematical inputs to the LP model. The inputs below correspond numerically to the requirements above. The notation corresponds to that of Figures 3 and 4. The point of this example is that the application builder states requirements in English and that the protocol designer reexpresses them mathematically. It is not important that the reader of this paper grasp the details of the mathematical expressions that follow. Protocol designer translation:

1.  $\sum_j a_{ij} x_j \leq b_i$  for  $p_1 \leq j \leq q_1$   
(Meaning: Consider only data transfer features.)
2.  $c_j = c_j + 0.2c_j$  for  $p_2 \leq j \leq q_2$   
(Meaning: Increase the value of these response features by 20 percent.)
3.  $x_j = 1$  for  $j = \dots$   
(Meaning: Indicate that certain features are required.)
4.  $c_j = c_j - 0.2c_j$  for  $p_3 \leq j \leq q_3$   
(Meaning: Decrease the value of these features by 20 percent.)
5.  $x_j = 0$  for  $j = \dots$   
(Meaning: Omit certain features from consideration.)
6.  $b_1 + b_2 + \dots + b_k \leq 30000$   
(Meaning: Constrain developmental resources to \$30,000.)
7.  $b_{k+1} + b_{k+2} + \dots + b_m \leq 0.05$  of present operational cost  
(Meaning: Constrain operation resources to 5 percent growth.)

##### *Parameters of the model*

The toy FTP design highlights some of the ways in which the model can be controlled. Presently, the model allows the user to constrain a number of feature and resource attributes. They appear below along with an indication of the actions that can be affected by the model's user.



### Feature attributes

Feature subsets of  $X$  can be selected from the data base for purposes of considering or not considering them. Also, their values and/or costs can be increased or decreased. The selection mechanism is to name one or more properties of the features to specify a subgroup of interest. These feature attributes are described below.

1. Features may be *selected as the subgroups* shown in Figure 4. For example, data transfer or host accounting may be constrained as subgroups. Thus, as a subgroup the features can be considered or ignored and their associated values can be increased or decreased.
2. Within a subgroup or over all features, *user options* of service capabilities may be addressed.
3. Similarly, *system responses* to events is an addressable attribute.
4. The presence or absence of *preassigned default values* for protocol features may also be addressed.
5. Lastly, the ability for the user to *set default values* is an attribute of protocol features that can be addressed.

### Resource attributes

1. The allowable allotment of resources may be managed on an individual resource basis, i.e.,  $\sum a_{ij}x_j \leq b_i$  for a given  $i$ .
2. Resources may be grouped as shown in Figure 4, e.g.,  $b_i + b_{i+1} + \dots + b_k \leq$  allotment.
3. Lastly, resources may be partitioned at the level of operational ones and developmental ones.

### Present work on a realistic FTP design

To date we have concentrated on developing the model. Consequently, it has been subjected only to small problems similar to the toy problem described above. Presently, we are constructing the data base needed for a genuine exercise. For this more substantial test, we have chosen the FTP family. To acquire features for the data base, the extraction rules<sup>11</sup> were applied to six FTP documents.<sup>2,3,4,8,14,18</sup> Due to the newness of these rules, the level of granularity of what we refer to as a feature varies somewhat. Generally, though, applying the rules yields a rather fine level of description of the protocol. Approximately 170 unique features were taken from the six FTP documents. We somewhat arbitrarily selected as the kernel FTP those features appearing in at least five of the six documents. The resulting kernel contains eight features, leaving about 160 for inclusion in the data base. For resources we have used the list shown in Figure 4 and we have provided initial estimates of costs, later to be replaced by those of protocol designers. We plan to work with several other government agencies to develop optimum FTPs to then be evaluated in terms of these agencies actual requirements of existing needs. These agencies will provide the initial value vectors for the data base.

## THE MODEL STRUCTURE AND IMPLEMENTATION

### Branch-and-bound algorithm

A linear representation was selected for our initial study of protocol design quantification. It is not clear that a linear relationship holds. Nevertheless, we emphasize the use of an analytical method to quantify protocol design. The correctness of this particular model is of lesser importance at this juncture.

One important characteristic of the model is that it is a binary integer model—binary in the sense that either a feature is present or absent in resulting protocol design. A number of (binary) integer programming algorithms have been studied in terms of their computational efficiency. Most of them can be classified as using enumeration, decomposition, cutting-plane, or group theoretic algorithms.<sup>6</sup> Initially, an enumerative branch-and-bound algorithm,<sup>12</sup> was chosen. The branch-and-bound method greatly reduces the number of possible solutions that need to be considered. It consists of a branch portion that partitions the search space and a bound portion that determines the best possible value (in the subspace) of the objective function. Useful insights were also gleaned from Mitten<sup>17</sup> and from Lawler and Wood.<sup>16</sup> The algorithm as first implemented, described by Hiller and Lieberman,<sup>12</sup> was not perfectly suited to our needs. We are currently modifying the algorithm to reduce computation time, given the extra information inherent in the problem definition.

### Sensitivity analysis

In linear programming, all of the model's data are assumed to be correct. Actually, values are estimates.<sup>12</sup> We are reasonably confident of the cost of each resource (supplied by protocol designers), but the value of features (supplied by potential users) should be less accurate initially. To assure near optimal solutions, the model is to be parameterized for sensitivity analysis. Sensitivity analysis allows investigating different values for certain parameters and determining the extent to which data can be changed while retaining an optimal solution.<sup>7</sup>

### The computer program

The LP model is implemented in the C language and runs under the UNIX<sup>®</sup> operating system on a PDP-11/45. At present, the data base and program are separate files that are loaded together as an on-line submitted batch job with terminal output. We are constructing a command interpreter as an interface so that the model's user can interactively adjust input parameters and make temporary modifications to the data base.

With the advent of a 160-feature data base, a severe problem arises. The search space for an optimum solution is  $2^N$  where  $N$  is the number of data base features for the family of interest. Although the branch-and-bound algorithm sig-

nificantly reduces (compared to an exhaustive enumeration) the number of possible solutions, the number  $2^{160}$  is still very large. We have taken two steps to circumvent this computational explosion. The FTP protocols will be developed section-wise, e.g., connection features, then data transfer features, and so forth. Thus, if features are partitioned into, say, 10 sections of about 16 features each, then the search space is reduced from  $2^{160}$  to  $10 \cdot 2^{16}$ . Also, we are exploring other heuristic algorithms that will quickly produce a good upper bound on the objective function. A successful procedure will allow the features to be regrouped into larger clusters for simultaneous consideration.

## OTHER USES OF THE MODEL

### *Ancillary outputs*

The most prominent output of the LP model is the feature list of the target protocol. The model fashions other useful information such as resource requirements. Resource requirements and their costs are output for each selected combination of features. Reference material is to be incorporated at the feature level. This material (which will become part of the program output) will report, for each target protocol feature, prevalent implementations and implementation strategies, document references to existing or proposed implementations of the features, and other information to guide implementation of the protocol.

We have referred to the LP model as aiding in protocol design, whereas the main output is a service feature list, more closely allied with a specification than a design. However, the implementation guide information lies more in the domain of design than specification. In fact, the LP process lies somewhere between specification and design.

## CONCLUSIONS

The LP model does not provide a completely satisfactory protocol design tool. It attempts to quantify *sufficiency* of service. Other equally important variables for later study are *quality of service* and the *interface representation* of the service to the user.

The model has been described to a number of protocol users and designers, and has been well received. Our expectation, as a result of this work, is that some variant of

this quantification technique will impact the development of the next generation of network protocols.

## REFERENCES

1. Boies, S. J., "User Behavior on an Interactive Computer System," *IBM Systems Journal*, Vol. 13, No. 1, pp. 2-18, 1974.
2. Butscher, B. and Heinze, W., "A File Transfer Protocol and Implementation," *Computer Communication Review*, ACM, Vol. 9, No. 3, pp. 2-12, July 1979.
3. Chorn, G. E., Christman, R. D., and Klingner, C. T., "The Standard File Transport Protocol," Los Alamos Scientific Laboratory, LA-7388-MS, August 1978.
4. Harry Forsdick and Alex McKenzie, *FTP Function Specification*, Bolt Beranek and Newman Inc., Report No. 4051, August 1979.
5. Frederiksen, J. R., "Survey of the State-of-the-Art in Human Factors in Computers," Science Applications, Inc., Arlington, VA, SAI-75-533-WA, 1975.
6. Geoffrion, A. M. and Marsten, R. E., "Integer Programming Algorithms: A Framework and State-of-the-Art Survey," *Management Science*, Vol. 18, No. 9, pp. 465-491, May 1972.
7. Geoffrion, A. M. and Nauss, R., "Parametric and Postoptimality Analysis in Integer Linear Programming," *Management Science*, Vol. 23, No. 5, pp. 453-466, January 1977.
8. Gien, Michel, "A File Transfer Protocol (FTP)," *Computer Networks*, Vol. 2, pp. 312-319, 1978.
9. Heafner, John F., Yonke, Martin D., and Jeffrey G. Rothenberg, "Design Considerations for a Computerized Message Service based on Washington, D.C. Navy Personnel," USC/Information Sciences Institute, Marina del Rey, Ca., ISI/WP-1, May 1976.
10. Heafner, John F., and Miller, Larry H., "Design Considerations for a Computerized Message Service Based on Triservice Operations Personnel at CINCPAC Headquarters, Camp Smith, Oahu," USC/Information Sciences Institute, Marina del Rey, Ca., ISI/WP-3, September 1976.
11. Heafner, John F., Nielsen, Frances H., and M. Wayne Shiveley, "Toward the Extraction of Service Features from Definitive Documents on High-Level, Network Protocols," to appear in *NCC Proceedings*, 1980.
12. Hiller, Fredrick S., and Lieberman, Gerald J., *Operations Research*, Holden-Day, Inc., San Francisco, Ca., 1974.
13. International Data Corporation, *The Domestic Computer Installation Data File Coding Manual*, Waltham, Ma., December 1978.
14. *A Network Independent File Transfer Protocol*, prepared by the High Level Protocol Group, IFIP, International Network Working Group, HLP/CP (78) 1, December 12, 1977.
15. "Reference Model of Open Systems Interconnection," International Organization for Standardization, ISO/TC97/SC16 N227, August 1979.
16. Lawler, E. L. and Wood, D. E., "Branch-and-Bound Methods: A Survey," *Operations Research*, Vol. 14, No. 4, pp. 699-719, 1966.
17. Mitten, L. G., "Branch-and-Bound Methods: General Formulation and Properties," *Operations Research*, Vol. 18, No. 1, pp. 24-34, January-February 1970.
18. Neigus, N. J., "File Transfer Protocol and Standards," in *ARPANET Protocol Handbook*, Feinler, Elizabeth and Jonathan Postel (eds.), Network Information Center, SRI International, January 1978.



# Extracting service features from protocol documents\*

by JOHN F. HEAFNER, FRANCES H. NIELSEN

National Bureau of Standards  
Washington, D.C.

and

M. WAYNE SHIVELEY

Defense Communications Agency  
Washington, D.C.

## INTRODUCTION

This paper describes a manual procedure for itemizing the service features of a computer network protocol. The procedure has been tested and refined several times. The testing is described here, along with analysis of test results and general observations. The authors conclude that, although the procedure now provides only partially satisfying results, such a method is needed to aid in quantifying users' protocol needs.

It is easy to envision rather diverse applications that, in turn, occasion quite different network protocol services. For example, remote job entry and file transfer are most handily accomplished over a logical virtual circuit, whereas transaction-oriented processing applications are better suited to a datagram type service. The requirements can differ (dependent upon the application) within a protocol layer. Similar to the variant types of connections needed, the above examples differ in their requirements for message or data segment sizes. It follows that if the using process views a protocol as a collection of service capabilities, then the collection needs vary across application classes. Thus, in alliance with the ISO reference model, [ISO], we subscribe to the idea of a family of protocols within each layer of the network protocol hierarchy, where different members of the family (i.e., individual protocols) provide different mixtures of services concomitant with the using processes requirements.

If we accept the premise that protocol designers do not have a firm grasp on application requirements, and that needs vary according to application, then we can accept the need to develop and apply methods to quantify the service capabilities of a protocol.

## *Need for a repeatable procedure for determining service features*

To quantify the service capabilities of a protocol we must first agree on exactly what service features of protocols are. Interesting as it may seem to those not working in the area of high level network protocols, at a recent protocol workshop there was debate and disagreement over exactly what the service features were of a particular protocol in question. The experts do not necessarily agree. Thus, the first quantification step is to devise a method of determining service features of a protocol.

It is desirable to study proposed protocols as well as in-place ones. The definitive source common to the two is usually a protocol specification document written in English and accompanied by sundry notations. That is generally the most one can expect from a proposed protocol. With this in mind, we can now state our goal more precisely. *Given a protocol specification document written in English, devise a set of rules to extract service features. This method must be repeatable.* That is, if some number of protocol designers derive features of a given protocol, they should extract essentially the same features. We say essentially because what is involved is the application of an algorithm to English text. History has shown us in other areas of computer science that we should expect only limited success. Thus, our goal is to arrive at a workable set of rules, not a perfect set.

Once features can be identified, then other analytical methods [HEAJ] can be used to associate features with applications.

## *First attempt at rules for feature extraction*

A small, cohesive set of rules for feature extraction have recently been defined. They have been applied to excerpts of the INWG File Transfer Protocol document [INWG] by some of our colleagues who are not experts in the area of

\* This work is a contribution of the National Bureau of Standards and is not subject to copyright.

network protocols. The results of these exercises have been studied and subsequently fallacious rules have been amended. These are matters to which we now attend.

## DESCRIPTION OF EXERCISES

### *Rules and baseline features*

Keeping in mind the aim of determining and quantifying application requirements of protocols, we began to devise feature extraction rules by posing questions about protocol service features.

1. What are the characteristics that a service feature must possess?
2. How do we go about extracting service features from a protocol specification?
3. How do we know we have listed them all?

Consider these questions in reverse order. First, we cannot know when we have listed all the features. A reasonable objective is to specify a systematic way of listing features such that the procedure can be reliably repeated by different protocol experts to arrive at almost the same feature set. In partial answer to the second question, most of the service features stem directly from the commands and parameters afforded the application process. For example, Sections 4-6 of the INWG File Transfer Protocol, which describe commands and parameters, would be the appropriate parts of that document. We couched the answer to the first question above in terms of how we might expect a respondent in a survey to quantify the sufficiency of a service feature. Thought of in these terms, we then had some guidance from survey questionnaire construction methods that permit quantification of a subject's response.

From the above reasoning and guidelines from survey research methods [BABE], we devised a set of extraction rules and then selected the INWG FTP as a reference point for their application in order to test their effectiveness. After the first of two exercises the rules were improved based upon results obtained. The present version of the rules are shown below.

### *Rules for extracting protocol service features from a protocol definition document*

**NOTE:** The following rules are given in precedence order. Thus, first apply Rule 0, Rule 1, Rule 2, Rule 3, Rule 4 and finally, apply Rule 5. Service feature as used in these rules means an individual service provided by a protocol to the using process. The lack of some characteristic is not considered a feature; therefore nothing should be listed if a rule does not apply.

**Rule 0:** Scan the entire protocol document to familiarize yourself with it before applying the following rules.

**Rule 1:** Features of a protocol are described in those parts of a protocol definition document containing an explanation

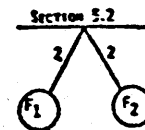
of protocol functions. Different terminology may be used in each document to designate these sections. For example, in the INWG File Transfer Protocol document, features can be extracted from those sections that describe commands and their parameters. In another protocol document, Transmission Control Protocol (TCP), the section containing the needed information is called Functional Description. *Use only those portions of the document that describe the actions or attributes of the protocol that are visible to the using process.*

**Example 1:** From the INWG File Transfer Protocol, Section 4 (...Level 0 Commands), Section 5 (Parameters Describing Standard Attributes), and Section 6 (Level 1 Commands) would be appropriate for feature extraction.

**Rule 2:** An individual feature may be described anywhere within a particular section of the protocol document. Sometimes two or more features can be extracted from a single description. It may be obvious, by the use of AND in the command description, that two features are being described (example 2); it may not be as obvious when a series of features is contained in a single command description. Also, there can be a description listing several related attributes which, when grouped together, constitute one type of service capability and therefore, one feature. In any case, where more than one feature is given simultaneously, *state each feature separately. If only one feature is described, state the feature.*

**Example 2:** From the INWG File Transfer Protocol, Section 5.2: "The mode of access parameter is used to define the DIRECTION and MODE of access to be used in the proposed transfer." The feature related to DIRECTION could be stated as: "Files may be transferred in either direction over a logical connection." The feature related to MODE could be stated as "The initiating process can select from among optional modes of file access."

We can represent this rule application graphically, where the arcs show rule numbers applied and the nodes portray the resulting features.



Remember that individual features can be listed in a series of three or more as well as being described singly or combined by the use of the conjunction AND.

When there is a description defining several parameters which can be grouped together by their related action, it may be helpful to examine the detailed bit patterns of the command formats to determine these groups. This approach to the application of Rule 2 is useful only where two conditions are met: (1) formats are included in the description, and (2) the difference between bits defining independent "concepts" and a field of related bits defining options of a single "concept" can be distinguished.

**Rule 3:** (Remember to apply Rule 2 before applying Rule 3. Then apply Rule 3, separately to each feature derived from Rule 2.) Often there are options or alternatives which can be applied to each action provided by the protocol. These alternatives may appear in either of two forms: options that a user may choose from in employing the feature, or a set of permissible responses, one (or more) of which a responding process may select. The features identifying the general service provided have been extracted by the application of Rule 2. Now apply Rule 3 to each of these features.

**Rule 3-a:** State, as a separate feature, the list of options or responses allowed. This rule may be applied recursively. That is, where there is a subcategory of options or responses associated with an option, state, as a separate feature, the list of these subcategory options or responses.

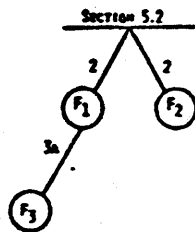
**Rule 3-b:** State, as a separate feature, that a default value is provided.

**Rule 3-c:** State, as a separate feature, the particular value of the default.

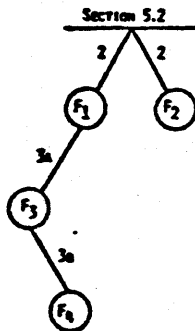
**Rule 3-d:** State, as a separate feature, the ability to change the value of the default.

Example 3: From the INWG File Transfer Protocol, Section 5.2, features could be stated as follows:

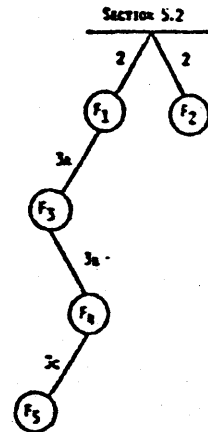
**Application of Rule 3-a:** "The initiating process can select access modes from among the following: 'make only', 'replace only', 'replace or make', 'append only', 'append or make', 'take job input', 'take job output', 'read and remove', 'read only', 'destructive read', 'give job input', 'give job output', and 'no access'."



**Application of Rule 3-b:** "There is a default access mode."

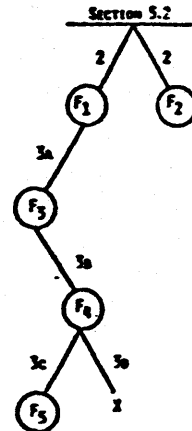


**Application of Rule 3-c:** "The default access mode is 'no access'."



**Application of Rule 3-d:** Access mode defaults cannot be changed in the INWG File Transfer Protocol; thus, according to the introductory note, no feature would be stated. If, however, the INWG File Transfer Protocol permitted the user to set the default, then the feature could be stated as: "The user may change the default access mode for the data transfer phase."

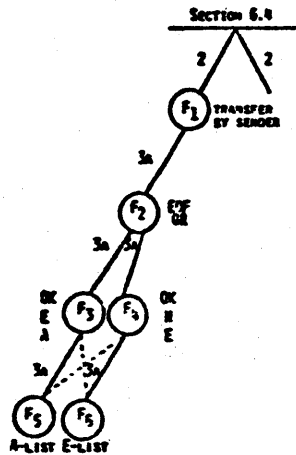
The x terminating the arc 3d indicates that no feature resulted from the application of the rule.



Example 4: From the INWG File Transfer Protocol, Section 6.4, which describes the End of Data Command (ES): a list of responses is given to signal the end of file. Beyond that, two of these responses ES(E)—error and ES(A)—abort, have optional responses associated with their use. Therefore, the following features could be listed:

**Application of Rule 3-a (applied recursively):** After the first application of Rule 3a to Section 6.4, we would expect the following feature to be extracted: "The ES command is sent in response to a QR or an EOF." A second application of Rule 3-a yields: "Possible arguments for an ES command sent in response to an EOF are OK, error and hold. Possible arguments for

an ES command sent in response to a QR are OK, error and abort." Another application of Rule 3-a gives the following: "The optional responses for the ES(Error) argument are 'receiver error, retry possible', 'receiver error, no retry possible', 'protocol error detected by receiver', 'GO not acceptable to receiver', 'sender error, retry possible', 'sender error, no retry possible', 'protocol error detected by sender', and 'GO not acceptable to sender.' "The optional responses for the ES(Abort) argument are 'timed out awaiting an MR', and so on.



**Rule 4:** Define each feature using a single declarative sentence.

Example 5: From the INWG File Transfer Protocol, Section 5.2: "Bit 15 (or [8000]) of N is used to indicate the direction of the proposed transfer, thus:

```
[0000] P→Q (TAKE)
[8000] Q→P (GIVE)"
```

This feature might be stated as follows: "Files may be transferred in either direction over a logical connection."

**Rule 5:** State features completely, clearly, and succinctly.

#### Baseline features by authors

For purposes of comparison to results of the exercises to be described, the authors applied the above rules to Sections 5.12, 5.13, and 5.14 of the INWG FTP. The service features thus obtained, we refer to as baseline features; they are given below.

#### SECTION 5.12

1. The user is able to optionally compress level 2 data.
2. The options for compression are to compress or not to compress level 2 data.
3. There is a default compression option.
4. The default is not to compress data.

5. The user may optionally interrupt and continue data transmission.
6. In doing so, the following options may be employed: later resumption of this transfer in a new transfer possible, restart requests permitted within this transfer, restart mark acknowledgment must occur, and temporary hold.
7. There is a default assumption.
8. The default is that no interrupt/restart will be used.
9. The user can provide parameters on the GO command for experimentation.
10. There is a default assumption.
11. The default is that the experimental facility will not be used.

#### SECTION 5.13

12. The service may inform the user of the current state of file transfer.
13. The states of file transfer are: viable, rejected, terminated, and aborted.
14. There is a default for the state of transfer.
15. The default is viable.
16. The user is provided an explanation when the transfer has failed or is being rejected.
17. The specific extra information about this state is given in Appendix IV of INWG FTP.
18. There is a default explanation.
19. The default value is given in Appendix IV.

#### SECTION 5.14

20. The user specifies a filename, which is acceptable to the conceptual filestore, that is used to identify the file involved in the file transfer.
21. There is a default filename.
22. The default filename is null.

Figure 1 is a graphical representation of the baseline features showing how they were derived from the FTP materials by applying the rules. The nodes represent the features where the number in the circle corresponds to the baseline feature number. The arcs are labeled to show which rule number was applied to derive the node below. The X means that the application of the rule did not result in a derived feature.

#### Exercises to test the rules

Exercises were devised whereby subjects would extract features from excerpts from INWG FTP in order to test the strength of the rules. Six colleagues, who were at least familiar with some high-level network protocols, were chosen to participate in the first exercise. They were divided randomly into two groups of three subjects each: a control group who would list service features, but without the aid of the extraction rules; and a test group who would also list service features described in the FTP material, while following the

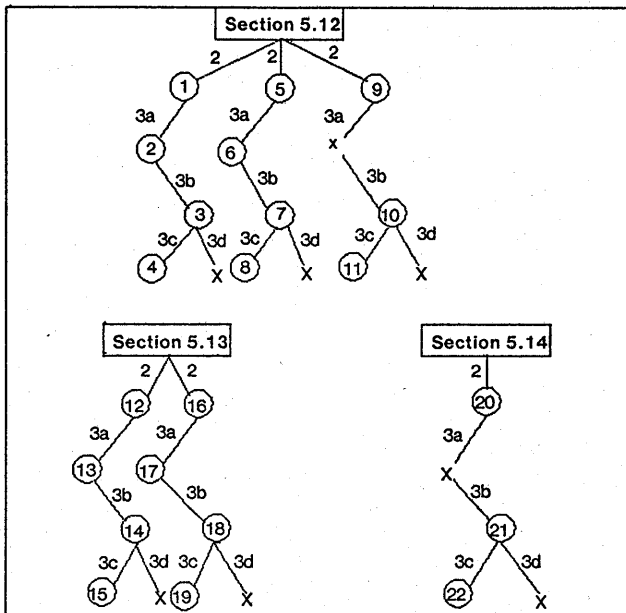


Figure 1—Representation of baseline features and application of extraction rules.

extraction rules. If our assertion that rules are needed is correct, then the control group members should each list different features. Also if the rules are good, then the test group members should list many of the features comprising the baseline. Pictorially, we can represent our expectations as shown in Figure 2. Responses from control group participants should be widely scattered, while the test group's responses should be similar among members and should roughly correspond to the baseline.

Results of the first exercise, to be described, led to rule improvement. Subsequently, a second exercise was conducted with six other colleagues, using the improved rules. The first part of the second exercise followed the pattern of the first exercise: control group and test group. Then, to gather additional information, the control group of the sec-

ond exercise later applied the rules to behave as a third test group. The results and conclusions follow.

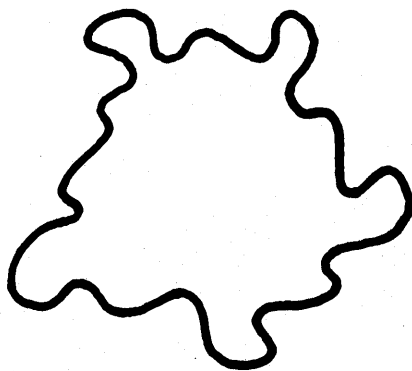
## FINDINGS AND CONCLUSIONS

### *Purpose of analytical analysis*

The test subjects of the first exercise named, respectively, 14, 7, and 7 of the 22 baseline features. The control subjects of that exercise identified, respectively, 2, 3, and 4 of the 22 baseline features. How successful was the exercise? Let us rephrase this question. More specifically, do the results warrant refining and continuing the initial approach, or instead do they suggest that this approach should be abandoned in favor of some new method? Furthermore, if the results are promising, what can be inferred from them that would lead to rule improvement?

These questions suggest the need to apply an analytical technique to aid in the evaluation of the exercise. Intuitively, we wish to group the subjects and the baseline data into groups such that subjects within a group have a high degree of natural association among themselves (according to some grouping criterion) while the groups themselves are more distinct from one another. Then, if the rules worked well, we would expect to derive three distinct groups: one containing the control subjects, a second containing the test subjects, and a third (closely related to the second) containing the baseline, as illustrated earlier by Figure 2. A statistical technique that can perform this grouping is called cluster analysis. Each subject (measured by 22 baseline features), along with the defined baseline, may be represented by a point in 22-dimensional space, whose coordinates for each subject are the subjects' scores associated with the 22 baseline features.

*Note that the results of use of the cluster analysis apply only to the 12 subjects of the two exercises and the baseline data. Based upon only 12 subjects, we cannot infer that they represent any larger population.*



**CONTROL GROUP**



**TEST GROUP**



**BASELINE**

Figure 2—Anticipated clustering of subjects' responses.



### Cluster model

The first step in the cluster analysis is to score the subjects' responses in relation to the baseline definition. If subject  $i$  named baseline feature  $j$  then the subject receives a score of 1 in that spatial dimension, otherwise the subject receives a 0 for that coordinate. It is important to recognize that assignment of scores is necessarily subjective, to some extent. The authors deliberated on scoring and concur that any errors that may have been induced through scoring would not significantly alter the results of the cluster analysis.

The next step in the analysis is to select a measure of association appropriate to the problem domain. Metric distance was chosen, since other common forms (e.g., cosine of the angle between vectors and product moment correlation) are considered to be ill suited to measuring associations with binary data [ANDM]. Common distance metrics used in cluster analysis are the  $l_1$ ,  $l_2$  (Euclidean), and  $l_\infty$  norms. With binary variables, each of these metrics compute to the same quantity for each comparison of subjects  $j$  and  $k$  on feature  $i$ . The distance metric, then, is used to compute pairwise relationships between subjects. The final major step in the analysis is to select and apply a clustering algorithm. The Ward algorithm [WIGD] was chosen because it weights the distance between centroids (as a function of cluster sizes) when computing the distances between clusters. Furthermore, the error sum of squares does not decrease, hence the algorithm is not subject to reversals.

### Cluster findings of exercise 1

If the rules were ideal we would expect to find two clusters: one containing the control group and a second containing the test group and baseline. When the 6 subjects are analyzed we find:

$$(c_1, c_2, c_3)(t_1, b)(t_2, t_3)$$

where the parenthetical groups represent clusters and the  $c_i, t_i$  and  $b$  correspond to control subjects, test subjects, and baseline, respectively.

The three clusters are roughly equidistant. Any further grouping results in a large increase in the accumulated error. Dividing any one group into two groups does not substantially reduce the error term. Hence, these are the appropriate clusters. The first cluster contains exactly the control group as we had hoped. The two remaining clusters contain (1) two test subjects, and (2) a test subject and the baseline.

Seven meaningful observations leading to rule improvement were made upon examining the cluster findings of exercise 1. The rules were accordingly modified (to the form shown earlier in this paper) for the second exercise.

### Observations and conclusions of a general nature from exercise 1

In addition to the rule-specific observations, four observations of a more general nature were noted.

1. *Observation:* Poorly written documents may result in nonmeaningful features extracted.

*Conclusion:* Rules may lead to standardized formats for writing specification documents.

2. *Observation:* Rules, as applied by the subjects, generate many features that are not of interest to users. An example is a feature dealing with parity bits. Most users would not care to be concerned—they would simply want parity to be such that their processes worked correctly.

*Conclusion:* Experts can act as filters to discard extraneous features, especially after some experience in interviewing applications users.

3. *Observation:* Control group subjects identified very few, yet very different, features and described them in general terms.

*Conclusion:* Such a high level of generality is not useful in obtaining detailed requirements from users. There is a need for rules to allow uniform feature extraction.

4. *Observation:* The cursory instruction period was inadequate.

*Conclusion:* Training needs to be extended and to include a work session by the subject with comments by a trainer.

### Overview of the second exercise

The first exercise allowed a comparison between control and test groups which we now refer to as  $C_1:T_1$ . The second exercise, conducted in two parts, permits the following comparisons.

$C_2:T_2$ : The control group 2 can be analyzed with test group 2. The purpose, as in  $C_1:T_1$ , is to test the need for extraction rules. In this respect, significant differences between exercise 1 and exercise 2 were the following. Control subjects of group 2 were instructed to state features at the level of individually distinguishable service capabilities. Thereby, they were given some level of training (which reflected the level of granularity of features) beyond that provided to control group 1. The rules were improved between exercise 1 and 2 which should improve the scores of test group 2 over test group 1. The training for test group 2 was more comprehensive than for test group 1 as suggested by general observation 4, above.

$T_1:T_2$ : Test groups 1 and 2 can be analyzed together. The purpose of this analysis is to determine the rule improvement's progress between exercise 1 and exercise 2.

$T_2:CT_2$ : As the second step of exercise 2, control group 2, after completing their initial exercise, was trained and then applied the rules. We now label this group control/test group 2,  $CT_2$ . Control/test group 2 can be analyzed with test group 2 to determine the impact of individual differences among the subjects (since the sample size is so small) with respect to the experiment. Note, however, that some factors of this comparison jeopardize the validity of results, namely the effects of taking a test upon the scores of a second test and maturation such as test fatigue.

$C_2:CT_2$ : We may analyze control/test group 2 and control group 2, i.e., the same group with and without the assistance of the rules. The purpose of this comparison is again to assess the value of the rules.

The results of these analyses follow.

*Findings of exercise 2*

The cluster analysis procedure is iterative. At each step exactly two groups of individual subjects who are nearest neighbors are combined. This synthesis introduces an error in that the resulting cluster is represented as the centroid of its components. Individual location information is lost through combining. Figure 3 graphs this accumulated error against repetitive clustering. We chose to describe the experiment results from the viewpoint of four iterations, which provides a cutoff prior to where the accumulated error doubled at the next iteration.

Based upon four iterations, findings follow. Results of exercise 1 are included for comparison. Notation is consistent with previous sections of this paper.

$C_1:T_1$	(c,c,c)	(t,t)	(t,b)
$C_2:T_2$	(c,c,c)	(t,t,t)	(b)
$T_1:T_2$	(t <sub>2</sub> )	(t <sub>1</sub> ,t <sub>1</sub> ,t <sub>2</sub> ,t <sub>2</sub> )	(t <sub>1</sub> ,b)
$T_2:CT_2$	(ct,ct,t)	(ct,t,t)	(b)
$C_2:CT_2$	(c,c,c,ct)	(ct,ct)	(b)

The number of baseline features identified by each member of each group is given below.

$C_1$ :	2, 3, 4
$T_1$ :	7, 7, 14
$C_2$ :	2, 2, 5
$T_2$ :	9, 9, 11
$CT_2$ :	4, 9, 12

*Interpretation of exercise 2 results*

The outcome of exercise 2 closely matches that of exercise 1. In aggregate, the correspondence to baseline features by subjects of exercise 1 were  $C_1=9$  baseline features and  $T_1=28$ . For exercise 2,  $C_2=9$  and  $T_2=29$ . The three-to-one 'hit' ratio between test group and control group maintained, as did the one-to-one ratio between  $C_1:C_2$  and  $T_1:T_2$ . Test group 2 was slightly more consistent than test group 1: compare (t,t,t) (b) to (t,t)(t,b). The improved rules and additional training did not noticeably improve the results ( $T_1=28$ ,  $T_2=29$ ). The findings of exercise 2 support (nearly duplicate) those of the earlier exercise.

The  $T_1:T_2$  comparison reveals random clustering. Again consistency is demonstrated but no real improvement is visible.

The comparison  $T_2:CT_2$  should signal any substantial differences between the control and test groups of exercise 2. If  $T_2$  results match  $CT_2$  results then we can infer that the differences between  $C_2:CT_2$  and  $C_2:T_2$  are mainly due to the use of the rules rather than because of any differences

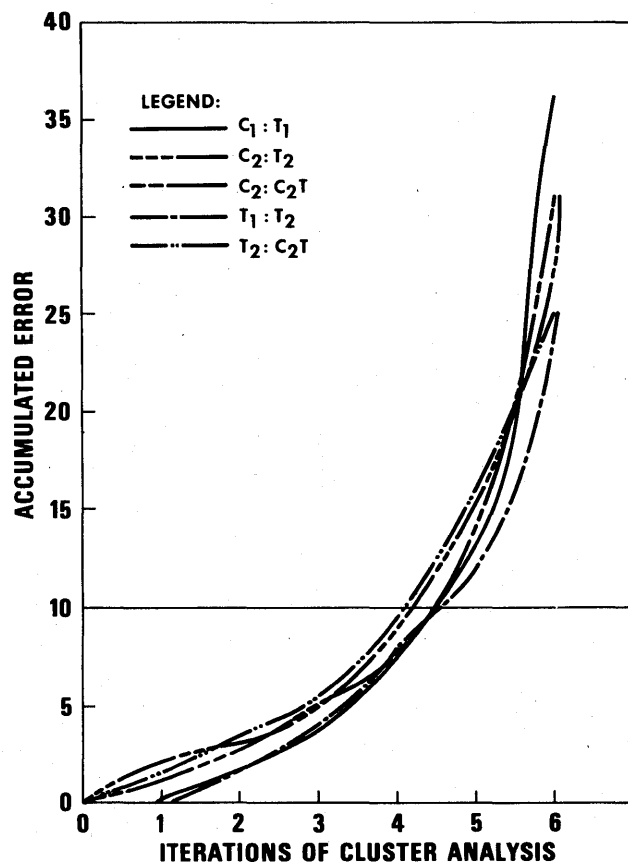


Figure 3—Error induced by grouping.

in the level of ability of subjects. The result (ct,t,t)(ct,ct,t) is ideal. We conclude that groups  $C_2$  and  $T_2$  were evenly matched overall and that the differences in scores can be attributed to the rules.

The results of comparison  $C_2:CT_2$  are similar to those of  $C_1:T_1$  and  $C_2:T_2$ , namely  $C_2=9$  and  $CT_2=25$ . The three-to-one ratio obtains. Evidently, the  $C_2:CT_2$  results reinforce the  $C_1:T_1$  and  $C_2:T_2$  findings, since after training the  $CT_2$  subjects performed similarly to the other test subjects.

*Other observations and conclusions from exercise 2*

Trained subjects (of both exercises) averaged naming 15 percent of the features of INWG Section 5.12. Their accuracy was 67 percent over Sections 5.13 and 5.14. The low hit rate for Section 5.12 was due to the misapplication of rule 3-a where rule 2 applied. Where rule 2 applies to features given in series, it should be distinguished from rule 3-a in that each feature expresses an independent concept.

Six of the 11 baseline features of Section 5.12 concern defaults. Misuse of rule 3-a caused features 1 and 5 to be combined into a single feature. Consequently, the subjects made two default statements instead of the six named in the baseline. Thus, they received scores of 0 for baseline features 1 and 5 and 3, 4, 7, 8, 10, and 11. This scoring may

seem rather strict. A more liberal interpretation of their default features would yield a 70 percent accuracy rate for Section 5.12 material.

Overall, test subjects of  $T_1$ ,  $T_2$ , and  $CT_2$  consistently identified slightly more than 40 percent of the baseline features.  $T_1$  received a 5-minute training.  $T_2$  and  $CT_2$  received a 45-minute training. The authors have worked with the rules over a several months period. We have independently applied the rules to six FTP documents. Upon comparing results we extract about 95 percent of the same features.

We conclude that the intent of the rules is basically sound. They need to be stated more crisply. They should include more examples to differentiate rules 2 and 3a. Lastly, with the present rules, considerable practice (on the order of days, not minutes) is probably needed to achieve a 95 percent hit rate with the baseline features.

#### NEXT STEPS

Given a repeatable procedure for feature extraction, the following question naturally arises. Do the features, so extracted, really represent the protocol's capabilities or value-added service in terms meaningful to the applications builders and to protocol designers? To answer the question with respect to protocol designers, the authors expect to get experts' opinions on their representativeness. The authors have since applied the resulting rules to several protocol documents and, with the help of applications users, are testing the results via an analytical model [HEAJ].

This effort provides insight that could be applied to the development of procedures for writing specifications. The actual identification of features reflects both the efforts of the author and the reader of the document. If the author had known the specific rules that were to be applied in the extraction of features, then an equivalent set of rules could have been identified and applied in the development of the document. It is anticipated that this dual approach would significantly enhance the elimination of ambiguity in such a document. There are plans to evaluate the utility of this approach in the development of standard protocol specifications.

#### REFERENCES

- [ANDM] Anderberg, Michael R., *Cluster Analysis for Applications*, Academic Press, 1973.
- [BABE] Babbie, Earl R., *Survey Research Methods*, Wadsworth Publishing Company, Inc., Belmont, Ca., 1973.
- [HEAJ] Heafner, John F. and Nielsen, Frances H., "A Linear Programming Model for Optimal Computer Network Protocol Design," to appear in *NCC Proceedings*, 1980.
- [INWG] *A Network Independent File Transfer Protocol*, prepared by the High Level Protocol Group, IFIP, International Network Working Group, HLP/CP (78) 1, December 12, 1977.
- [ISO] *Reference Model for Open Systems Architecture*, International Organization for Standardization, ISO/TC97/SC16 N117, November 1978.
- [TCP] *Transmission Control Protocol*, TCP Version 4, Information Sciences Institute, University of Southern California, IEN: 81, February 1979.
- [WISD] Wishart, David, "An Algorithm for Hierarchical Classification," *Biometrics*, Volume 22, Number 1, pp. 165-170, March 1969.

# Verification of information in a file\*

by JAINENDRA K. NAVLAKHA

Florida International University  
Miami, Florida

## INTRODUCTION

Computer centers and particularly data processing centers deal with a stack load of information files as a daily routine. The information in many of these files is organized according to a particular format. For example, in an "employee file," the information about an employee might consist of the name of that employee, his or her age, sex, social security number, salary, etc., in some order.

Before getting information about an employee from that file, it would be worthwhile to check the correctness of such information, that is, to make sure that in the "name" field is the person's name (some alphameric characters with blanks), in the "age" field is the age of the employee (that is, a two digit number specifying the age in years), and so on.

Observe that we are not validating the information for any particular employee as such, but making sure that the information available in the file is formatted properly. For example, if a number "25" appears in the "age" field, the contents of the field are okay from the format point of view even though the actual information might be wrong. However, if the age-field contains "A7," we are certain that this is wrong information from the format point of view.

Thus, what we wish to accomplish is the validation of information in the file inasmuch as it conforms to some format specification.

## BASIC VERIFIER SYSTEM

Figure 1 describes the verifier system, which when input with a proper valid format and a file record, verifies whether the information in the file record is correct with respect to the specified format or not.

The system is implemented on the UNIVAC 1106 at the Centro de Procesamiento, "Dr. Arturo Rosenblueth" in Mexico City, in Regular Expression Compiler-Markov (REC/MK) language. A couple of compilers of REC are described in Cisneros<sup>1</sup> and McKintosh.<sup>2</sup>

\* This work was carried out when the author was an exchange visitor to the Centro de Procesamiento, "Dr. Arturo Rosenblueth" ("Dr. Arturo Rosenblueth" Computer Center) in Mexico City.

REC/MK has no unconditional jump statements as some of the other programming languages have, but it does have an "escape statement" similar to the one in BLISS.<sup>3</sup> In that regard so far as its structure is concerned, this programming language falls somewhere between the structured language in the sense of Dijkstra, and an unstructured language like Fortran IV. The choice of the implementation language was governed more by the usage of the verifier at the Center rather than the suitability of the language for this particular application.

There are two major advantages of such a verifier system.

### 1. Reliability

Once the contents of all the records of a file are verified correct with respect to the proper format specification, one's confidence in the future usage of data from that file increases to a great extent.

### 2. Savings in programmer time

Without such a generalized system, if one wishes to verify the correctness of information content in a file, one has to write a separate program to achieve this. If there is a second file in which the information is organized quite differently from the previous file information, then yet another program is required to verify the correctness of information in that file. However, if a generalized verifier system did exist, then the information contents of both the files would be verified by it just by a change in format specification for the two files, thereby providing a tremendous savings in the programmer time.

## FORMAT TYPES

In order to check the correctness of information in a file record, the verifier takes the description of the valid format which specifies the order in which the information should occur in that record. This valid format is composed of a collection of different format types, which specify the type of information in fields of a file record. The choice of selecting particular types of formats was governed by the applications of the verifier system at the Center, though we feel that these formats are quite essential for verification of

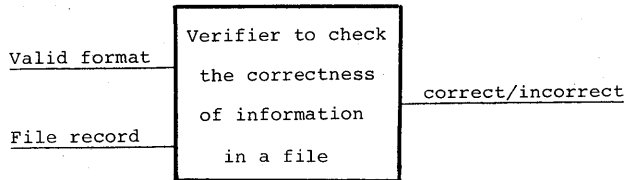


Figure 1—The verifier system

file contents in other application areas also. Quite a number of these format types are based on defining sets of characters, for example, 'D' is treated as a set of digits, 'L' as a set of letters, 'B' as a set of blank, etc. In the present version of the system, the following format specifications are allowed.

1. Digit-format

The digit-format allows one to specify the requirement that the next *n* characters in the file record should be all digits. It is specified as "nD," where *n* is a positive integer. Consider the following file record

-----1 5 7 9 3 b b A X Y ----- \*\*  
 |  
 1

where pointer 1 indicates the position in a file record to the left of which the information has been verified, and the verification is to be continued to its right. Suppose, the format is "5D." Then, the information in the next field of the file record is correct with respect to this format, and the pointer 1 in the file record is moved right by 5 characters.

2. Letter-format

Syntactically, it is specified as "nL" and means that the next *n* characters in the file record should be all letters (A to Z).

3. Ambos (means "Both" in English)-format

This format is specified as "nA" and requires each of the next *n* characters of the file record to be either a letter or a digit.

4. Blank-format

It is specified as "nB" and it looks for *n* blanks in the information file record.

5. Total-format

"nT" specifies that the next *n* characters in the file can be any characters chosen from the full character set under consideration. This format allows one to skip over several characters in the file.

\*\* b denotes a blank character.

6. Set definition

This format allows the user to define a set *X* of particular characters, so that he can specify "Set format" (described next) based on this set.

example 1— $X = L + (\cdot b)$  defines *X* to be the set  
 $X = \{A B \dots Y Z \cdot b \}$

This set might be useful when verifying characters in the name-field of an employee file record.

$X = L - (XYZ)$  specifies a set *X* whose members are all the letters of the alphabet except 'X', 'Y', and 'Z'.

$X = T - (+ - * /)$  defines a set of all characters (recall that *T* is the total set) except '+', '-', '\*', and '/'.

We feel that this format is one of the most essential formats for information verification in a file. There are many occasions when such a format can be used to check the contents of a particular field in a file record. One example might be the verification of name-field contents of an employee file, when the set can be as defined in example 1.

7. Set-format

"nX" specifies that the next *n* characters in the file should belong to the set *X* defined using the set definition format.

For example, consider the format

$X = L + D - (12YZ) \$ 5 X \$$

where '\$' is the separator between two formats. It first defines the set  $X = \{3 4 \dots 9 0 A B \dots W X\}$  and then looks for 5 characters in the file record to belong to the set *X*. Obviously, the set definition precedes the set format specification.

example 2—Suppose a record of an employee file has the following set up.

Field Name	NAME	AGE	SEX
No. of Characters	20	2	1

Assume that we wish to leave two blanks between any two adjacent fields. For verifying the correctness of information in this record, the valid format might be specified as follows:

$X = L + (\cdot b) \$ 20 X \$ 2 B \$ 2 D \$ 2 B \$$   
 $X = (MF) \$ 1 X \$$

If a record was . . .

JOHNbP.bDOE,JR.bbbbbb27bbM

then, it would be verified correct with respect to the above format specification.

8. Constant value format

On a number of occasions, one wishes to specify a sequence of specific characters in a file. This can be done using

the constant value format. It is specified as "C**b** (<char.sequence>)" where (<char.sequence>) is the desired sequence of characters in the file. '\$' acts as the terminator of the desired sequence, and thus cannot be used as part of the sequence itself.

For example, C**b**CONSTANT\$

requires the next eight characters in the file to be "CONSTANT."

9. Range-format

Syntactically, this format appears as R(<char>-<char>, <char>-<char>,....). It thus consists of a variable number of character pairs of the type <char>-<char> each being separated from the next by ','. The *i*th character pair governs the information content of the *i*th character in the file.

R(0-6,A-K,X-Z), for example,

looks for three characters in the file, the first between 0 and 6, the second between A and K, and the third between X and Z. The obvious restriction of the format is that in every character pair, the ASC11 (or any other) representation of the first character should be less than or equal to that of the second character.

All the above formats, except the Set definition format, were fixed-length field formats which are character oriented. The formats explained below are a little different from the above types.

10. Number-format

It is specified as N(<digit seq>, <digit seq>). It is used when the information in the next field of the file record is desired to be within a range of numbers. For example, while verifying the "age" field of a record in the employee file, one may wish to make sure that the age is between 21 and 65. The format specification for that information would be N(21,65).

Both the numbers specifying the range should be positive (current implementation, which can easily be changed to include negative numbers, also) and the first should be less than or equal to the second if the format is to be meaningful. In the present version of the verifier, there is an added restriction imposed on this format specification which is that the two digit sequences would have the same number of characters. For example, a number between 0 and 379 should be specified as N(000,379). This restriction can be removed in future versions of the verifier.

11. Select-format

This format allows the user to specify information content of the next field in the file to be one of many choices. It is specified as follows:

S(choice 1 ! choice 2 ! choice 3 ! ..... ! choice *k*).

For the next field information of the file to be correct, it should satisfy either one of the above choices, e.g., S(C**b**M!C**b**F) specifies that the next character in the file should be either an 'M' or an 'F'. This format may be useful for verifying "sex" field of an employee file record.

A choice in this format specification can be either of Digit, Letter, Ambos, Blank, Number, Range and Set formats or Constant value format.

For example, S(3**D** ! 1**B** ! 3**L**) looks for the field information to be consisting of either 3 digits or 1 blank or 3 letters. The file record

----- 1 **b** + 37 -----  
          |  
          1

satisfies the above format because the second choice, 1 *B*, is satisfied.

12. Variable-format

There are times when we know that the next field contents should be all digits or all letters or all characters from some set, but we do not know the size of the field. In fact, the size of the field may be different for different records of that file. Variable-format proves to be very useful under those circumstances. It allows the user to specify a variable length information in the file. The syntax of this format is ...#special letter, where special letter can be *D*(Digit), *L*(Letter), *A*(Ambos), *B*(Blanks) or *X*(Set). It means that any number of next characters in the file belonging to the particular set (specified by special letter) are to be counted as constituting the next field. Absence of any character in the file from the designated set means finding 0 character.

For example, if the file information is

----- 1 *A B X + Y\** -----  
          ↑  
          1

then, #*D* matches 1 character,  
      #*L* matches 0 character,  
      #*A* matches 4 characters,  
      #*B* matches 0 characters,  
      and if  $X = L + D + (+)$ , then  
      #*X* matches 6 characters.

One application of this format might be to skip over some variable number of characters in the file until you encounter the first '\*'. That could be accomplished by the following format specification.

$X = T - (*) \$ \# X \$$

First, *X* is defined to be a set of all characters except '\*', and then, in the file you match a variable number of characters belonging to the set *X*.

13. Equal-format

It is not uncommon to encounter file records where the information content of two fields should be identical. For

example, the identification code in a file may appear as the first and also the last field of the file record. Equal-format validates such information.

It is specified as "Em," where  $m$  is the number of one of the previous fields.

For example, If the valid-format is ...

3D \$ 2B \$ E1 \$ - - - - -

it looks first for three digits, then for two blanks and then the third field, whose contents should be the same as those of the first field, character by character. The file record...

- - - 15266152-----  
 ↑  
 1

satisfies the above three formats of the valid-format.

By convention, when counting field in this format specification, set definition is not counted as a field because it does not verify any additional field of the file record.

Figure 2 gives the formal BNF description of a "valid-format" for verifying information content of a file record.

## HIGH-LEVEL DESCRIPTION OF THE PROGRAM

The verifier takes in the description of a valid format as input and then reads in a sequence of file records to be verified as specified by individual formats of the (valid-format).

Once it finishes verifying one set of file records, it goes back and reads in another (valid-format) and repeats the procedure until no more verification is necessary. The algorithm in its simplest form can be described as follows:

- Step 1: Read a (valid-format). If "end-of-input-file," then STOP.
- Step 2: Read a file record to be verified. If no more records to be verified, GO TO step 1.
- Step 3: Validate the information of the file record according to (valid-format), and report the result. GO TO step 2.

Step 3 of this simple algorithm checks whether the information in successive fields of the file record is conformal with respect to the individual formats of the (valid-format) or not. Its detailed description is given below:

- Step 1: Get the next (individual format).
- Step 2: If DONE, then perform some initialization for the next record verification, and RETURN.
- Step 3: Verify the information in the next field of the file record.
- Step 4: Report the correctness information to the user. GO TO step 1.

Step 3 above consists of first determining what the type of that (individual-format) is and then verifying the contents of the next field with respect to it.

```

<Valid format> ::= <ind. format> $ |
                <ind. format> $ <Valid format>
<ind. format> ::= <D-for> | <L-for> | <A-for> |
                <B-for> | <T-for> | <Set-for> |
                <Set def.> | <Con Val> | <N-for> |
                <R-for> | <S-for> | <V-for> |
                <E-for>
<D-for>       ::= <pos. int.> D
<L-for>       ::= <pos. int.> L
<A-for>       ::= <pos. int.> A
<B-for>       ::= <pos. int.> B
<T-for>       ::= <pos. int.> T
<Set-for>     ::= <pos. int.> X
<Set def.>    ::= X = <Set description>
<Set description> ::= <LS> | <LS> + <Set description> |
                    <LS> - <Set description>
<LS>          ::= <sp. let. 1> | <set>
<sp. let. 1>  ::= D | L | T
<set>         ::= (<char. sequence>)
<char. sequence> ::= <char> | <char.> <char. sequence>
<char>        ::= any valid character
<pos. int.>    ::= <nonzero digit> |
                    <nonzero digit> <digit sequence>
<digit sequence> ::= <digit> | <digit> <digit sequence>
<digit>        ::= 0 | <nonzero digit>
<nonzero digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<Con val>     ::= C <Blank> <char. sequence>
<Blank>       ::= ␣
<N-for>       ::= N (<digit sequence> , <digit sequence>)
<R-for>       ::= R (<Range>)
<Range>       ::= <char> - <char> |
                    <char> - <char> , <Range>
<S-for>       ::= S (<choices>)
<choices>     ::= <ind. choice> |
                    <ind. choice> ! <choices>
<ind. choice> ::= <D-for> | <A-for> | <L-for> |
                    <B-for> | <Set-for> | <Con val> |
                    <N-for> | <R-for>
<V-for>       ::= # <sp. let 2>
<sp. let 2>   ::= D | L | A | B | X
<E-for>       ::= E <pos. int.>
  
```

Figure 2—BNF description of a valid-format

A sample output of the program for a test run is shown in the APPENDIX.

## CONCLUSIONS AND EXTENSIONS

In this paper, we described a software verifier system to check the correctness of information in a file. The (valid-format) and the file record to be verified are input to the

system, and the output is the correctness result for every (individual-format) of the (valid-format), based on whether the file information was conformal with respect to it, or not.

Other than increasing one's confidence in the contents of the file once it is verified, the system, being a rather general one, helps save a tremendous amount of programmer time in the sense that one program is good for verification of information in all types of files and separate programs are not required.

The program is written in REC/Markov language and is implemented on the UNIVAC 1106 computer at the Centro de Procesamiento, "Dr. Arturo Rosenblueth" in Mexico City. The choice of the language was governed by the future use of the verifier at the Center rather than its suitability for this application. The top-down design of the system has guaranteed modularity, and we feel that the system can be maintained and modified fairly easily.

A couple of possible extensions to the system are proposed below:

1. Presently, set definition allows only 'X' to be used for defining sets. This can be easily changed to include usage of other characters ('Y' and 'Z,' for example) in set definitions. Then, more complex set definitions would be possible and it would be possible to keep 2 or 3 defined sets at hand. Also, this would allow sets to be defined in terms of already defined sets other than D, L, or T.

For example,  $Y = X - Z$

2. Except for equal-format, all other types of formats are independent of information contents of previous fields of a file record. For more complex applications, a few other "information dependent formats" might be added.

For example, a format of the type...

IF information in kth field was\*\*\*\*\*,  
THEN format 1 ELSE format 2 \$ ,

may be quite useful in certain applications. At present, this is partially achieved by the select format.

One other possibility that comes to mind is as follows. Suppose  $S(C\text{b}M!C\text{b}F)$  was pth individual format. Then a format of the type...

IF "contents of field p of the file conformed with CbM" THEN format 1 ELSE format 2 \$

may allow one to tie in proper information in some field of the file record with contents of one of the previous fields whose contents might have been one of many choices.

3. Presently, the verifier system is capable of handling only the "character oriented files." This can be extended to make it handle binary strings of more than one byte.

I would like to add that one can visualize tremendous possibilities of such a verifier system in data file information verification. This, being the first version of this system, is definitely not very neat and concise, but with proper modifications, the later version of the system can be developed to satisfy the needs of individual users.

## ACKNOWLEDGMENTS

This work was carried out when I was an exchange visitor at the Centro de Procesamiento in Mexico City. My sincere thanks to all the personnel of the Department of Systems Support and Department of Computer Graphics, particularly, Carlos Garcia, Genaro Mariscal, Rafael Garcia and Rafael Tello.

## REFERENCES

1. Cisneros, Gerardo, "A Fortran Coded Regular Expression Compiler for the IBM 1130 Computing System," *Acta Mexicana de Ciencia y Tecnologia*, Vol. 4, No. 1, 1970.
2. McIntosh, Harold V., "A CONVERT Compiler of REC for PDP-8," *Acta Mexicana de Ciencia y Tecnologia*, Vol. 2, No. 1, 1968, pp. 33-43.
3. Wulf, W. A., Russell, D. B. and Haberman, A. N., "BLISS: A language for systems programming," *Comm. of ACM*, Vol. 14, No. 12, Dec. 1979, pp. 780-790.

## APPENDIX

A sample output of the verifier (test runs) is shown below.

### REC/MARKOV

INPUT A FULL FORMAT

FULL FORMAT

3D \$ 2B \$ 3L \$ 2B \$ N(173,257) \$ 2B \$ CbXYZ\$ 2B \$ 3A \$

INPUT A FILE RECORD

FILE RECORD BEING TESTED =

123 ABC 197 XYZ A9L

FIELD NO. = 1

CORRECT INFORMATION

FIELD NO. = 2

CORRECT INFORMATION

FIELD NO. = 3

CORRECT INFORMATION

FIELD NO. = 4

CORRECT INFORMATION

FIELD NO. = 5

CORRECT INFORMATION

FIELD NO. = 6

CORRECT INFORMATION

FIELD NO. = 7

CORRECT INFORMATION

FIELD NO. = 8

CORRECT INFORMATION

FIELD NO. = 9

CORRECT INFORMATION

INPUT A FILE RECORD

FILE RECORD BEING TESTED =

086 AOR 258 X-Z WIT

FIELD NO. = 1

CORRECT INFORMATION

FIELD NO. = 2

CORRECT INFORMATION

FIELD NO. = 3



CORRECT INFORMATION  
 FIELD NO. = 4  
 CORRECT INFORMATION  
 FIELD NO. = 5  
 ERROR ((N-FOR) VIOLATED)  
 FIELD NO. = 6  
 CORRECT INFORMATION  
 FIELD NO. = 7  
 ERROR ((CON-VAL) VIOLATED)  
 FIELD NO. = 8  
 CORRECT INFORMATION  
 FIELD NO. = 9  
 CORRECT INFORMATION

INPUT A FILE RECORD (\*DONE CASE\*)  
 INPUT A FULL FORMAT  
 FULL FORMAT  
 $X = L + D - (12YZ) \$ 3X \$ 2B \$ S(2X | R(0-5,0-6) | 2B)$   
 $\$ 5T \$ 2B \$ 2X \$$   
 INPUT A FILE RECORD  
 FILE RECORD BEING TESTED =  
 S5M 15JKNAV K7  
 SET DEFINITION  
 CORRECT INFORMATION  
 FIELD NO. = 1  
 CORRECT INFORMATION  
 FIELD NO. = 2  
 CORRECT INFORMATION  
 FIELD NO. = 3  
 CORRECT INFORMATION  
 FIELD NO. = 4  
 CORRECT INFORMATION  
 FIELD NO. = 5  
 CORRECT INFORMATION  
 FIELD NO. = 6  
 CORRECT INFORMATION

INPUT A FILE RECORD  
 FILE RECORD BEING TESTED =  
 TRY N -\*/+? 6U  
 SET DEFINITION  
 CORRECT INFORMATION  
 FIELD NO. = 1  
 ERROR ((SET-FORMAT) VIOLATED)  
 FIELD NO. = 2  
 ERROR ((B-FORMAT) VIOLATED)  
 FIELD NO. = 3  
 CORRECT INFORMATION  
 FIELD NO. = 4  
 CORRECT INFORMATION

FIELD NO. = 5  
 CORRECT INFORMATION  
 FIELD NO. = 6  
 CORRECT INFORMATION

INPUT A FILE RECORD (\*DONE CASE\*)  
 INPUT A FULL FORMAT  
 FULL FORMAT  
 $\#A \$ 2B \$ E1 \$ 3D \$ E4 \$ 3L \$ E6 \$$   
 INPUT A FILE RECORD  
 FILE RECORD BEING TESTED =  
 A79BX A79BX159159AXTAXT  
 FIELD NO. = 1  
 CORRECT INFORMATION  
 (5 CHARACTERS MATCHED)  
 FIELD NO. = 2  
 CORRECT INFORMATION  
 FIELD NO. = 3  
 CORRECT INFORMATION  
 FIELD NO. = 4  
 CORRECT INFORMATION  
 FIELD NO. = 5  
 CORRECT INFORMATION  
 FIELD NO. = 6  
 CORRECT INFORMATION  
 FIELD NO. = 7  
 CORRECT INFORMATION

INPUT A FILE RECORD  
 FILE RECORD BEING TESTED =  
 JKN124 JKN124173172A56A56  
 FIELD NO. = 1  
 CORRECT INFORMATION  
 (6 CHARACTERS MATCHED)  
 FIELD NO. = 2  
 CORRECT INFORMATION  
 FIELD NO. = 3  
 CORRECT INFORMATION  
 FIELD NO. = 4  
 CORRECT INFORMATION  
 FIELD NO. = 5  
 ERROR (UNEQUAL FIELD INFORMATION)  
 FIELD NO. = 6  
 ERROR ((L-format) VIOLATED)  
 FIELD NO. = 7  
 CORRECT INFORMATION  
 INPUT A FILE RECORD (\*DONE CASE\*)  
 INPUT A FULL FORMAT (\*DONE CASE\*)  
 REC/MARKOV.

# Translating non-standard extensions to standard Pascal

by VISWANATHAN SANTHANAM

Wichita State University  
Wichita, Kansas

## INTRODUCTION

Extensions to a programming language are often introduced by individual implementors for a variety of reasons. In some cases, a new construct may allow the programmer to take advantage of a unique hardware or software feature in the specific system. In other cases, the application for which the language is frequently employed may warrant simpler and more efficient constructs. Whatever the motive, extensions, in general, severely limit portability of programs. Often it is necessary to manually translate non-standard features before a program can be successfully adapted to a new environment. If we must live with such extensions at all, ideally we would like to be able to translate all extensions automatically to make the adaptation expedient and reliable.

Some extensions clearly do not have direct translations in all target environments. Others lend themselves to system-independent translations, e.g., to the base (standard) language itself. Typically such translations do not have the same elegance and efficiency as the original construct, but they enhance the portability of programs. A system which accepts descriptions of such extensions and their translations can automatically process non-standard programs, yield a standard translation and provide a precise definition of the extension to a naive reader of the program. In this paper, we outline such a system that we call "extension processor."

The details of the system are specific to the Pascal language. The concepts involved, however, are equally applicable to any block structured language, such as Algol, BCPL, and PL/1. Their applicability to other languages such as FORTRAN and COBOL is limited.

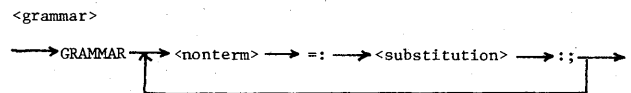
The extension processor is somewhat similar in function to a macro processor in that both systems attempt to preprocess the source text of a program and produce an output suitable for the compiler. There are a number of macro processors designed for use with Pascal-like languages (see References 3 and 4 for examples) and some of these are powerful enough to handle minor extensions such as the use of constant expressions where only simple constants are permitted in the base language. But these systems are not capable of describing more complex forms of translations that may involve source code movement and/or augmentations far from the point where the extension is used. The proposed extension processor can effectively describe moderate forms of

global and local transformations that are sufficient for translating many popular extensions to Pascal.

## THE EXTENSION PROCESSOR

Extensions are described by the programmer with the help of two new statements—GRAMMAR and TRANSLATE. The GRAMMAR statement describes the syntax of the extensions, while the TRANSLATE is used to specify the standard language translations for them. Each of these statements can be included in any Pascal block, preceding all other declarations such as LABEL, CONST, etc. Like all other declarations, GRAMMAR and TRANSLATE are also subject to the scope rules of Pascal. This means an extension can be "global" if defined at the program level, or "local" if defined in a lower level block. It is also possible to translate the same extension in different ways in different blocks. The complete syntax of these statements is outlined in the appendix. The overall structures and the semantics are discussed in this section.

The main structure of the GRAMMAR statement is depicted in the syntax diagram below.



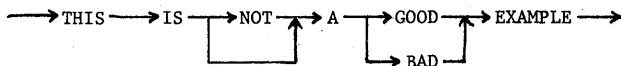
Here <non-term> is the syntactic identifier of the extension feature; <substitution> is merely a linear representation of a syntax diagram for that extension. The <substitution> itself is a sequence of tokens consisting of key words (both standard and user defined), operators and other non-terminals, defined elsewhere in a GRAMMAR statement. Not all non-terminals need to be explicitly defined; "standard" definitions could be included within the extension processor, just as standard type identifiers, such as INTEGER, BOOLEAN, etc., are defined in Pascal.

In addition to the terminals and non-terminals, a substitution may also include labels to identify syntactic entities in the definition. These labels are useful in describing the translation later in a TRANSLATE statement.

There are two special operators provided in the GRAMMAR statement: the "follow-on" operator → which is op-

tionally employed to clarify the physical juxtaposition of two successive syntactic entities; and the "alternates" operator -: which indicates a branch-off point in the syntax diagram. For example, the syntax diagram

<example1>



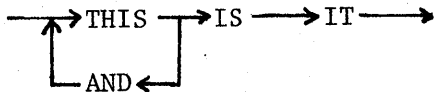
can be expressed in a GRAMMAR statement like this:

#### GRAMMAR

```
.example1. =: THIS IS -: .( NOT ),
                .( );
                → A -: .( GOOD ),
                .( BAD );
                → EXAMPLE ;;
```

Recursive definitions are written in a manner similar to the BNF notation. For example, this syntax diagram

<example2>



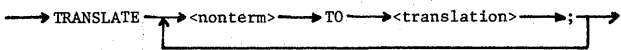
is translated to a GRAMMAR statement by defining a separate non-terminal for the recursive component "THIS":

#### GRAMMAR

```
.example2. =: .t. "see definition below"
                -: .( ), "the escape option"
                .( AND .t. ); "the recursion option"
                → IS IT ;;
.t. =: THIS ;;
```

The TRANSLATE statement is more complex as it must describe the more difficult task of translating the extension to the standard language. Many times there is a need to add variables or type identifiers to the local block or the program block. Sometimes it may also be necessary to add executable statements to the body of a procedure. The TRANSLATE statement syntax provides for these specifications in Pascal-like constructs. The overall syntax diagram for the statement has the form:

<translate>



The non-terminal referenced above must be defined in a GRAMMAR statement, either explicitly or implicitly as a standard non-terminal. The <translation> consists of three components—"global includes," "local includes" and "substitution." The global and local "includes" are intended to specify new labels, identifiers, procedures, functions or

statements that may be needed at the program level and in the current block respectively. Thus each of these "includes" is divided into eight parts: LABEL, CONST, TYPE, VAR, PROCEDURE, FUNCTION, TOPOFSTMT and BOTOFSTMT.

The syntax of each part, so far as the extension processor is concerned, is the same. However, the phrases constructed by the extension processor must be acceptable to the Pascal compiler. The actions of the extension processor itself are restricted to adding the phrases specified in each part to the appropriate parts of the program or local block.

The substitution part is identified by the key word TRANS and it describes the in-place substitution for the recognized extension. The description takes the form of a sequence of terminals (operators, key words etc.), non-terminal labels (defined in the GRAMMAR for the extension), and identifiers and labels defined in the "includes" (called "sysnames" and "syslabels"). Special built-in functions called "sysfunctions" are also provided for generating tokens that do not appear explicitly within the extension text, but are "computable" by the extension processor. For example, &EXPTYPE(\$E) generates the type of the expression \$E.

Furthermore, three powerful "syscontrol" statements are also provided for the control of the number and sequence of tokens generated; these are &FOR (also, &FOREACH), &IF and &CASE. These syscontrols are similar in function to the conditional compilation features of many macro processors.

The sysfunctions and syscontrols together form the heart of the translation language used by the extension processor. By adding more powerful features into these two categories, it is possible to widen the range of extensions that can be translated. In the following section, we illustrate the use of some of the more basic features of this work.

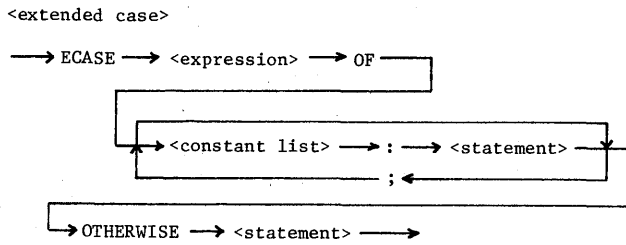
#### EXAMPLES

A set of five examples is presented in this section to illustrate features of the extension processor.

##### *Example 1: the extended CASE statement*

Many implementations of Pascal, such as Reference 1 for example, have extended the CASE statement to accept an "OTHERWISE" phrase which specifies an action for cases not covered in the body of the main CASE. This example illustrates how such a CASE statement can be translated to the standard language.

Even though it is not necessary, we will assume that the extended CASE statement begins with a different word, say ECASE. This change makes it easier to describe the extension as a new statement and the parser within the extension processor can recognize it more easily. With this modification the typical extended CASE construct can be described in a syntax diagram as follows.



This syntax is described in a GRAMMAR statement as follows.

```

GRAMMAR
.statement. =: -: (.extendedcase. ). "include as a
                statement"
                ;;
.extendedcase. =: ECASE $EXP: .expression. OF
                  $CASES: .cases.
                  OTHERWISE $OTHSTMT: .state-
                    ment. ;;
.cases. =: -: $NULL: .( ).,
              $CASE: .("non-null cases"
                $CONL: .constantlist. : .state-
                  ment. );,
              -: .( ).,
              .( ; .cases. ); "recurse"
                ;;
  
```

The labels such as \$EXP, \$CASES etc. are defined here for reference in the TRANSLATE statement.

A straightforward translation of the ECASE statement is to check the value of the case expression to be in the set of all constants for which case parts are defined before executing a standard CASE statement. The idea is expressed in a TRANSLATE statement as follows.

```

TRANSLATE .extendedcase. TO
LOCAL "additions to the current block"
VAR &X: &EXPTYPE.( $EXP ); "define a local
variable"
TRANS "in-line substitution"
BEGIN (* extended CASE *)
&X := $EXP; (* compute and save case
expression *)
IF &X IN
[ "now, list all constants"
&FOR $CASES &DO "begin looking at
$CASES"
.( &FOREACH $CONL &DO
.( $CONL "list the constants"
&IFMORE.( , ). "comma if more to go"
).
). ] "ending bracket for the set"
THEN (* if one of the constants *)
CASE $EXP OF "do a standard CASE"
$CASES "list the cases as is"
  
```

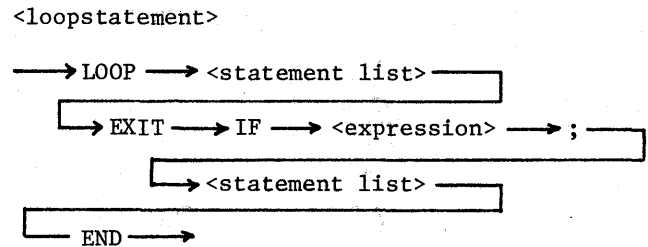
```

END (* CASE *)
ELSE $OTHSTMT "the otherwise part"
END (* extended CASE *)
TREND; "end translation"
  
```

The &FOR \$CASES is really not needed as there is only one \$CONL in the GRAMMAR for ECASE. It is included here for clarity that the inner &FOREACH is confined to the non-terminal \$CASES.

Example 2: the LOOP statement

The NBS Pascal compiler<sup>2</sup> introduces the notion of a LOOP statement to allow efficient construction certain classes of loops without the use of the GOTO statement. (The NBS compiler does not itself implement the GOTO.) The syntax of this extension is described by



and in a GRAMMAR statement by

```

GRAMMAR
.statement. =: -: (.loopstatement. ). ;;
.loopstatement. =: LOOP
                  $SL1: .statementlist.
                  EXIT IF $EXP: .expression. ;
                  $SL2: .statementlist.
                  END ;; "end of definition"
  
```

The translation of the LOOP statement is straightforward if the use of GOTO statements is permitted.

```

TRANSLATE .loopstatement. TO
LOCAL LABEL &2; "new label needed for GOTO"
TRANS
BEGIN (* LOOP *)
REPEAT
$SL1;
IF $EXP THEN GOTO &2; (* EXIT IF *)
$SL2
UNTIL FALSE; (* forever, that is *)
&2: (* point of escape *)
END (* LOOP *)
TREND;
  
```

This translation may be unacceptable if GOTOs are not implemented in the target system either. A translation without

the GOTO can also be devised, though somewhat belatedly.

```

TRANSLATE .loopstatement. TO "the goto-less version"
  LOCAL "define a local procedure for $SL1"
    PROCEDURE &SL1;
      BEGIN
        $SL1
      END;
  TRANS "now, use &SL1 for $SL1 and unfold the loop"
    BEGIN
      &SL1; (* leading execution of $SL1 *)
      WHILE NOT ( $EXP ) DO
        BEGIN
          $SL2;
          &SL1 (* top of the loop here again *)
        END
      END (* LOOP *)
    TREND;

```

#### Example 3: conditional expressions

Algol's conditional expressions can save coding time and improve execution speed of programs. Pascal does not include this feature in the spirit of keeping the language simple. We can readily introduce this feature for simple data types as an extension.

```

GRAMMAR
.expression. =: -: .( .conditionalexp. ); "add to
expressions" ;;
.conditionalexp. =: CIF "choose a different prefix for ease"
  $BOOL: .expression.
  THEN ( $EXP1: .expression. )
  ELSE ( $EXP2: .expression. ) ;;

```

We translate the entire conditional expression into a function call, which then returns one of the two values, \$EXP1 or \$EXP2, depending on \$BOOL.

```

TRANSLATE .conditionalexp. TO
  LOCAL
    FUNCTION &F : &TYPE.( $EXP ). ;
      BEGIN
        IF $BOOL THEN &F := $EXP1
        ELSE &F := $EXP2
      END;
  TRANS "the in-place substitution is simple"
    &F
  TREND;

```

#### Example 4: macro statements

The extension processor described in this paper does not necessarily replace the functions of a macro processor. A

simple text substitution type macro processor can express certain preprocessing requirements better than our extension processor can. The most logical place for a macro processor is between the extension processor and the compiler; that is, the output from the extension processor can be the input to the macro processor whose output in turn will be the compiler's input. This means that any non-standard construct that is to be passed on to the macro processor must be left alone by the extension processor. In particular, the MACRO statement itself must be acceptable to the extension processor. This can be achieved by stating the syntax of the MACRO statement in a GRAMMAR statement at the program level. An example of such a declaration follows.

```

GRAMMAR
.declarations. =: -: .( .macrodecl. ); "add to
declarations" ;;
.macrodecl. =: MACRO .macrodefs. ;;
.macrodefs. =: $MACNAME: .identifier.
  -: .( )., "case of no parameters"
  .( .idlist. ); "with parms"
  = # .terminals. # ; "prototype"
  -: .( )., "end of macros"
  .( .macrodefs. ); "more to go"
  ;; "end of MACRO syntax".

```

Additional definitions may be needed if the macro parameters are represented by special identifiers.

#### Example 5: the BUILTIN attribute

In PL/1 it is possible to return to the global level meaning of an identifier, even though a surrounding block may have redefined it differently. The BUILTIN attribute of PL/1, which does this, is somewhat restrictive, but the concept is quite useful. A similar facility could be made available in Pascal if we define a key word "BUILTIN" and allow declarations such as

```

TYPE BOOLEAN = BUILTIN;

```

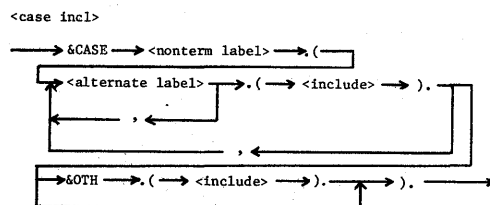
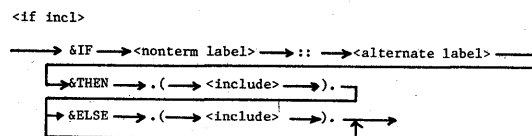
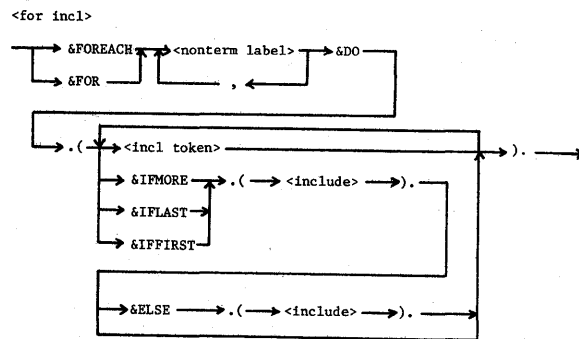
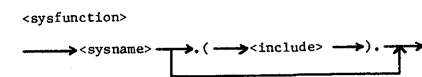
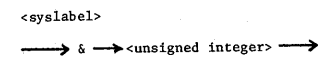
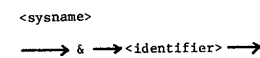
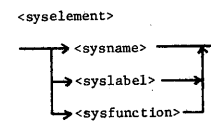
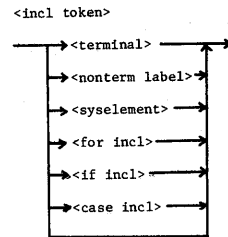
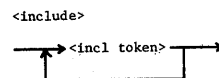
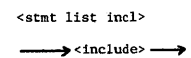
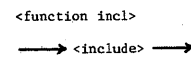
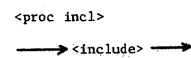
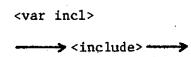
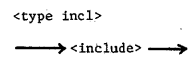
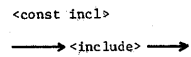
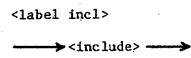
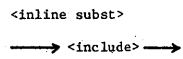
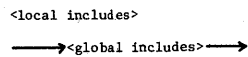
A similar use with the CONST declaration may also be permitted. The extension processor can be called in to convert this non-standard feature into standard Pascal by defining a new identifier at the program level, equating it to the standard identifier there and substituting the new identifier in place of BUILTIN. The following GRAMMAR and TRANSLATE statements achieve this.

```

GRAMMAR
.constphrase. =: -: .( .builtinphrase. ); ;;
.typephrase. =: -: .( .builtinphrase. ); ;;
.builtinphrase. =: $X: .identifier. = BUILTIN ;;
TRANSLATE .builtinphrase. TO
  GLOBAL
    TYPE &GLOBALTYPE = $X;
  TRANS
    $X = &GLOBTYPE
  TREND;

```





# The flexible console—FLEXICON

by DAVID L. STEINBERG

*Siemens Corporation*  
Cherry Hill, New Jersey

## INTRODUCTION

Mature engineering fields have well-developed tools and standards which aid in development and guarantee product quality. Ironically, after over thirty years, programming remains basically an undisciplined process with most software engineers still using techniques and tools which have changed little over the last ten years.<sup>1</sup> There is as yet no common standard for programming tools and common experience in the use of those tools and techniques which currently do exist. The mounting cost of software dictates that the time has come to offer the software engineer tools which can increase his productivity, via saving time and making man-machine interaction more efficient.

A set of programming tools, brought together in a software development facility such as a dedicated programmer's console, could provide the framework for a software engineering discipline, and add stability to the computer programming environment. Because of the continuing reduction in the cost of hardware it is now economically plausible to consider the design of a single user terminal with capabilities and features which could not have been justified in the recent past.

The project undertaken by CRD has examined the "wish list" indicated above with the aim of designing a workstation which allows the programmer to work efficiently with an integrated set of electronic tools in place of traditional tools such as pencil and paper, line editors, keypunches, documentation typists, draftspeople and printed reference manuals.<sup>2</sup>

The traditional stages of software engineering or programming (planning, implementation, and testing) were considered to be the activities performed by programmer in which electronic assistance is needed. The primary questions that arose were:

- What tools does a programmer need?
- Can these be satisfied by a programmer's console?

Several criteria were used to select those features to be included in the Programmer's Workstation.<sup>3</sup> These included: (1) the applicability of the features to the software effort; (2) the degree of usefulness of the feature to the software effort; (3) the potential power of the feature within the system environment; (4) the ease of implementation of the feature into the prototype.

As work proceeded on the project, it became apparent that many features being designed into FLEXICON could be effectively utilized in applications other than programming. Therefore, FLEXICON was designed as a basic system which could easily be adapted to tasks other than that of a Programmer's Workstation.

## SYSTEM CONFIGURATION OF THE FLEXIBLE CONSOLE

The hardware configuration of FLEXICON makes use of independent MPU subsystems sharing a common bus. Although up to three subsystems can be interconnected via a data link controller, only two were used in the engineering model: a dialog subsystem and a function subsystem. The dialog subsystem manages the man-machine interaction. This system includes an INTEL Single Board Computer (SBC-80/20) with 64 kBytes of memory with related boards and interfaces (see Figures 1 and 2).

The function subsystem performs tasks initiated by the dialog subsystem such as file management, spooling, etc. It is implemented by an INTEL Microcomputer Development System (MDS) with 64 kBytes of memory, double density dual floppy disks, an In-Circuit Emulator (ICE-80), a resident relocater and linker, a text editor, an operating system, and assorted utility routines.

An additional subsystem can be added to accommodate special requirements, for example archival storage or parallel high level language compilation.

Since the tasks a programmer performs require significant amounts of data, a high-speed data link between the dialog subsystem and the function subsystem was needed. Data exchange on the prototype was provided by Direct Memory Access (DMA) in each of the subsystems via a parallel data bus.

The various user I/O peripherals, which have been connected to the dialog subsystem, represent a collection of state-of-art devices intended to provide the user with an optimized man-machine interface. These include: (1) up to four color CRTs, each with limited graphics; (2) transparent Touch Sensitive Devices (TSD) on each CRT; (3) conventional keyboard.

The CRTs are Intercolor 8051 intelligent terminals, which can display 48 lines consisting of up to 80 characters per line.



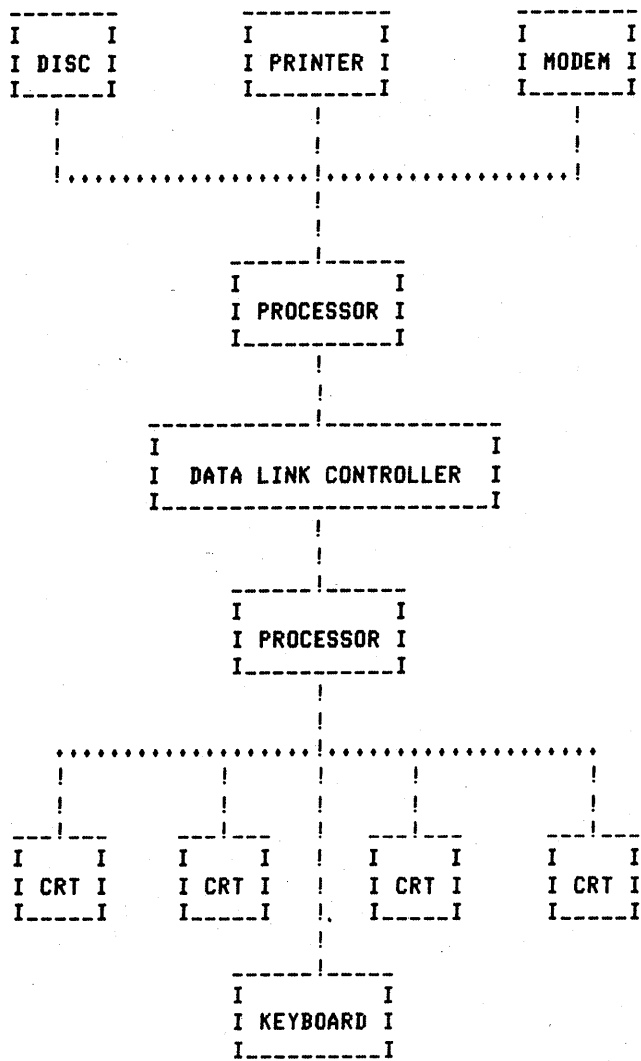


Figure 1—Hardware structure of Flexicon.

FLEXICON's use of CRT displays is intended to minimize the need for paper. In this respect, most tasks require a worker to have access to several kinds of information at the same time in order to make decisions and/or to create new documents. The number of sources of information a worker needs at any one time and the amount of information a worker needs to see from each particular source is totally task dependent. However, there is a practical limit to the amount of information that a person can physically perceive. The selection of four CRTs was done arbitrarily for the engineering model as an upper limit, with further study to determine an optimum configuration, taking into account ergonomics, applications, etc.

FLEXICON utilizes modes, control areas, as well as function "buttons" on each screen to organize the system with the objective of maximizing displayed information. CRD found that the basic capabilities required to organize the needs and tasks of a programmer appear to be similar to those required in many other application areas which involve

the creation and modification of data or information. CRD believes many of the concepts behind the use of FLEXICON as a programmer's workstation are appropriate in configuring FLEXICON as a station for users performing similar tasks.

With the display of more information on multiple CRTs than previously available, CRD believes that effective, "more natural" methods of interacting with this information must be devised to replace traditional cursor control on the keyboard, light pens, joy sticks, etc. Studies by N. Negroponte have shown that traditional cursor control methods significantly detract from an efficient interaction.<sup>6</sup> For example, a light pen requires a worker to be continually handling it as he shifts back and forth between entering information on the keyboard and using the light pen to manipulate information on CRT screens.

Two approaches have been used in FLEXICON to provide a more natural method of interacting with the large amount of displayed information. First, screen oriented techniques are used in place of the traditional line editor. In other words, the user "sees" everything as it actually is, without the need to imagine as in the line editor approach of the past. Each screen is considered to be a small window into a larger workspace belonging to the file. At all times, a FLEXICON user sees on a screen as much of the file's workspace as the CRT field allows (42 lines). What the user sees in the CRT window is an exact image of the information in the file workspace. A FLEXICON user can concentrate on the job he has to do instead of being distracted by traditional line editing tools which intercede between him and the information with which he is working. No clutter of commands ever appears in the window area of a FLEXICON CRT screen; only the user's information appears.

A second technique to provide a "comfortable" interaction with information displayed on the CRTs involves the utilization of transparent Touch Sensitive Devices (TSD) which cover the face of each CRT. The TSDs enable the FLEXICON user to manipulate information appearing in a CRT window simply by touching the information on the CRT

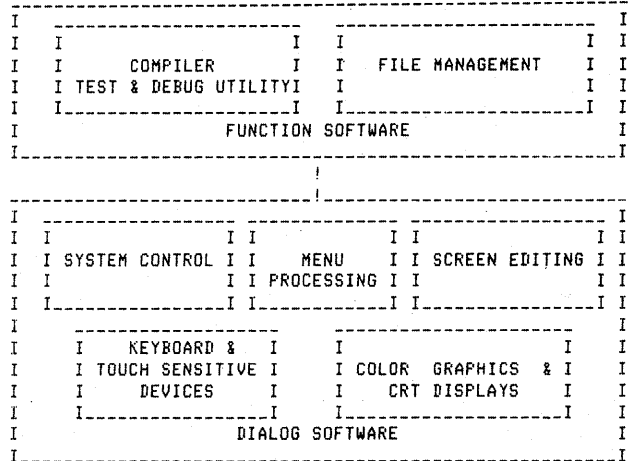


Figure 2—Software configuration of Flexicon.

with his fingertip, thus liberating the user from the inconvenience of handling multiple light pens and/or keyboards. The Touch Sensitive Devices (TSD), together with the visible workspace concepts just described, greatly simplify user interaction with FLEXICON and provide a comfortable and friendly user environment for man-machine interaction.

The use of high resolution graphics is considered not essential to document software. The capability to easily draw lines and boxes and color modification is thought to be adequate for program documentation. However, CRD feels that the graphics provided to the programmer must blend well with individual character manipulation capabilities since programmers create and edit diagrams made up of lines and boxes and alpha-numeric characters and since such diagrams are often intermixed with text.

FLEXICON provides a user with simple, yet adequate, graphics which enable him to create and edit diagrams made up of lines and boxes. An easy to comprehend and easy to use set of editing functions are built into FLEXICON which enables a user to modify, move and edit graphic lines and boxes, including any associated text. These editing functions are all natural language controls. A FLEXICON user can manipulate a portion of the CRT window either as a strip of characters or as a geometric area. The result of combining simple line graphics with both graphic and character oriented editing techniques is a system as easy to use as a ruler and a pair of scissors, but with the power and the convenience of electronic manipulation. All of the controls are able to be activated by touch, rather than by command entry via the keyboard.

Color is available as an added dimension to document clarity. Its use is being studied as a way to enhance various presentations, such as structured program code constructs.

## USING THE FLEXIBLE CONSOLE

Traditionally, clerical staff members have used terminals with limited capabilities to do repetitive, rigidly defined tasks. Special knowledge is required to properly use such terminals and to properly perform such tasks. Usually the clerical staff requires extensive training in the usage of a particular terminal and how to perform a particular task.

With the growth of timesharing computer systems, programmers are beginning to use terminals to create and revise programs. Most of these terminals are teletypes or dumb CRTs equivalent to teletypes. To use such terminals to create and edit programs, programmers must learn a command language for a line editor as well as separate command languages to perform such tasks as compilation, linking, program execution and file management. Usually these various command languages are designed for the convenience of computers and terminal equipment rather than humans.

A major design goal of the FLEXICON project has been to produce a unit which can be used after only a short introduction. FLEXICON does not require special knowledge or training to use and does not require the user to know any special command language. FLEXICON is controlled and

used by touching items on menus, by answering questions posed by the system, and by following directions given through the use of prompts.

In FLEXICON, standard menu buttons on the Touch Sensitive Devices (TSDs) surround each screen. These standard menu buttons are fixed onto the TSDs and do not change. These standard menu buttons organize the major tasks of the user and provide most of the tools he needs to do his job. The present FLEXICON system has been oriented toward a "Programmer's Workstation." Therefore, the standard menu buttons provide the tools to create and edit code, diagrams and documentation. These tools are similar to, and in some ways identical to, the tools needed by other professionals and by secretarial staff to create and edit information. Nevertheless, these menu buttons are really controlled and therefore may easily be changed to suit other applications.

In FLEXICON, "soft" menu buttons are presented to the user in the control area of the screen to provide appropriate functions for specific applications needs. These menu buttons are changed as necessary as part of the prompting scenario. They provide a simple and powerful technique to provide the user of FLEXICON with very specific capabilities tailored precisely to a particular application task. They activate frequently used preprocessed operations.

The Touch Sensitive Devices and the combination of standard, fixed menu buttons and "soft," changeable menu buttons make the FLEXICON unit a natural and simple tool. A FLEXICON user touches the information he wants to manipulate, thereby positioning the cursor, and then touches a menu button to perform certain action with the information.

In addition to the standard and soft menu buttons, FLEXICON uses informational prompts to guide and assist the user in performing tasks. The lower six lines of each CRT screen constitute a control area and are used by FLEXICON to constantly inform the user of the status of the screen including any errors he may have made and the action required to correct the error and properly proceed with his task. FLEXICON also asks questions of the user when it needs specific information to perform a task.

The result of using standard menu buttons, "soft" menu buttons, TSDs and extensive prompting on FLEXICON, is a terminal system concept which can be used productively by a new user in a very short period of time. A user need remember very little about the system to use FLEXICON. He only need look at the physical face of the screens with their associated fixed and soft menu buttons and the prompts to start to work.

## APPLICATION AREAS FOR FLEXIBLE CONSOLE

The specific application area of a "Programmer's Workstation" was selected for the initial FLEXICON system. However, the basic tools and capabilities a programmer needs to electronically create, modify, and use software and documentation, and to interchange information with other computer systems, are also the basic tools and capabilities needed by other professionals and staff who create, modify

and use information. FLEXICON is a basic core system to which additional capabilities and functions can be added to tailor the system to other application areas. It is appropriate to consider the use of FLEXICON in any application area in which professionals need user oriented facilities and the capabilities of a highly intelligent terminal to increase their productivity and job satisfaction.

Some of the specific areas in which FLEXICON seems to lend itself include:

- Management Information Systems
- Computer Aided Design
- Automated Architecture
- Database Management
- Database Inquiry
- Secretarial Workareas
- Automated Office
- Order Entry with Inventory Management
- Process Control

Generally, any tasks which can be structured and performed with decision trees, that require the integration of information from several sources or that require the accessing and/or combining of information from several sources are likely candidates for a system such as FLEXICON.

## CONCLUSIONS

An all purpose workstation which enables a programmer to increase his productivity, reduce the time required to develop a system and increase the quality of his work, offers advantages: to the programmer, the system designer, the analyst and, indeed, the manager. It must be noted that FLEXICON is not the solution to all a programmer's problems; there are several areas which need more development, specifically the application package itself. The FLEXICON

concept is that of a high level tool which can have many applications. There are many additions which can be made to this system to make it a more powerful tool. Much of these are software expansions. CRD is currently looking into various areas in which the computer may ease the load of the programmer to an even greater extent. Some of these areas include such dynamic possibilities as automatic programming, automatic testing, electronic manuals, automatic program analysis, and non-procedural languages.

The future for a system such as FLEXICON is open-ended, constrained only by the needs of the user and technology. The power of the system is defined by the memory size and the usage, be it as an intelligent terminal, a stand-alone system, or part of a network. FLEXICON could be tied to a micro, a mini, or main frame computer.

Computer based systems are being used more and more in all facets of life. Their successful use requires good man-machine interfaces. FLEXICON provides convenient, easy-to-use, man-machine interface, which speeds and simplifies the creation and manipulation of the information needed by these computer based systems.

## REFERENCES

1. Gaines, B. R. and Hill, D. R. (eds.), "Man Computer Communications," Infotech State of the Art Report, Infotech International, London, 1979.
2. Berinson, N., "FLEXICON, A Model of a Multi-Screen Programmer's Desk." Siemens Research and Development Reports (June 1978), ISSN 0370-9736, pp. 353-355.
3. Reed, D., et al., "FLEXICON User Specifications," Version 1.0, Siemens, Cherry Hill, 1978.
4. Ivie, E. L., "The Programmer's Workbench—A Machine for Software Development," *Communications of the ACM*, 20, 1977, pp. 746-743.
5. Weinberg, G., *The Psychology of Computer Programming*, New York, Van Nostrand Reinhold Company, 1971.
6. Negroponte, N., et al., "Argumentation of Human Resources in Command and Control Through Multiple Media Man-Machine Interaction," in *Graphical Conversation Theory*, ed. by Department of Architecture, Massachusetts Institute of Technology, Cambridge, 1976.

# The INTEL<sup>®</sup> 8087 numeric data processor

by JOHN F. PALMER

Intel Corporation  
Santa Clara, California

## INTRODUCTION

The INTEL<sup>®</sup> 8087 is a high performance general purpose numeric data processor. It is used with the INTEL<sup>®</sup> 8086, or the INTEL<sup>®</sup> 8088, microprocessors to extend their instruction sets with over 100 instructions (not counting addressing mode). The 8087 has all of the 8086 addressing modes and through a coprocessing interface is able to execute numeric instructions concurrently with the 8086 (or 8088). The high performance overlapped execution is transparent to the user who sees the 8087 simply as an extension of the 8086 (8088). Furthermore, the 8087 is the only chip that must be added to an 8086-based system to provide numerics capability with a performance enhancement over software of more than 100. In addition to high performance, great care was taken to ensure that the 8087 could be used in any application involving numbers—including commercial calculations. This required an unprecedented level of accuracy and reliability to be built into the processor. The intent was to greatly simplify the production of high performance but reliable numeric software.

Mathematical software is easy for the uninitiated to write but notoriously hard for the expert. This paradox exists because the beginner is satisfied if his code usually works in his own machine while the expert attempts, against overwhelming obstacles, to produce programs that always work on a large number of computers. The problem is that while standard formulas of mathematics are fairly easy to translate into FORTRAN they often are subject to instabilities due to roundoff error. Consider, for example, the quadratic equation

$$Ax^2 - 2Bx + C = 0$$

whose solutions are

$$x_1 = (B + \sqrt{B^2 - AC})/A$$

$$x_2 = (B - \sqrt{B^2 - AC})/A$$

Programs using these formulas, when run on a conventional computer, will produce results that are very sensitive to roundoff damage.

Since roundoff analysis is subtle, difficult and exceedingly tedious, our intent in the 8087 design was not only to make reliable and robust software easier for the expert to build

but to make it more likely that the unanalyzed code of the average programmer would run successfully. For example, the above formulas for the quadratic roots will be far less sensitive to roundoff error if evaluated on the 8087 instead of a typical computer.

Another important aspect of the 8087 is that it is an implementation of a very carefully designed standard, proposed to the IEEE and destined to be emulated by many other manufacturers. The establishment of this standard will go far to provide an environment for experts to produce ever more reliable software. Until now most experts, in an attempt to produce portable code, have written for a mythical computer whose capabilities are an intersection of the capabilities of all major computers and whose arithmetic is a collection of all the ugliness of any of them. Thus these programs, while useful for everyone, are ideal for no one. Assuming a standard environment, professional programmers will be able to concentrate on optimizing the code since portability will be automatic.

The proposed IEEE Floating-Point Standard (1, 2, 3, 4) specifies two data formats

REAL (32 bits, 8 bit exponent)

LONG REAL (64 bits, 11 bit exponent)

and a support format we call

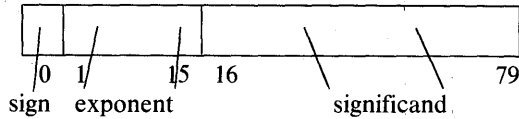
TEMPORARY REAL (80 bits, 15 bit exponent)

to indicate its intended use as a format to hold intermediate results. Along with the formats, the standard specifies three rounding rules, required operations (+, -, \*, /, REM, SQRT, COMPARE) and exception conditions. The 8087 implements the full standard and many extensions. Some of the major benefits provided by the 8087 will be explained shortly but first an architectural overview will be given to serve as a framework for the more detailed later discussion.

## ARCHITECTURAL OVERVIEW

The major architectural feature of the 8087 is its operand-result stack of 8 registers, each capable of storing an operand

in TEMPORARY REAL (80 bit) format as shown below:



All operands used within the 8087 are first converted to this format which provides 64 bits of precision and a range of about  $10^{\pm 4900}$ . In addition to the stack there is a set of registers called the ENVIRONMENT which contains the exception flags and pointers and processor control flags.

At the simplest level the programmer may treat the 8087 registers as a pure stack. All operands are explicitly loaded into the stack and operations are performed on the top elements of the stack. The load (or push) instruction can transfer operands to the stack using any one of seven data formats:

- Shorter Integer (16 bit 2's complement)
- Integer (32 bit 2's complement)
- Long Integer (64 bit 2's complement)
- Real (32 bit)
- Long Real (64 bit)
- Temporary Real (80 bit)
- Packed Decimal (80 bit; 18 digits and sign)

The load instruction never causes a rounding error, since TEMPORARY REAL is precise enough to hold all seven types exactly. Stack operands can be returned to memory in any one of these seven forms using the store and pop instruction which automatically converts the top of stack to the designated format, stores it in memory and then pops the stack.

The arithmetic operations which manipulate the stack pop the top two elements off the stack, perform the operation and push the result back onto the stack. The operations supported are: ADD, SUBTRACT, SUBTRACT REVERSE, MULTIPLY, DIVIDE, and DIVIDE REVERSE. There are COMPARE instructions that set two bits in the environment (indicating "greater," "equal," "less," or "unordered") and then pop both elements, pop just the top, or pop neither. The REMAINDER instruction in the 8087 is an instruction primitive. It is intended to be used in a software loop to return both the "divisor" and the partial remainder of a division. There are several other instructions that operate on the top elements of the stack:

- NEGATE:** reverses the sign of the top of stack
- ABSOLUTE VALUE:** sets the sign of the top of stack to positive
- SQRT:** computes the square root (its operation time is as fast as divide) of the top of stack
- SCALE:** treats the next-of-top as an integer and adds it to the exponent of the top of stack—a fast form of multiplying by a power of two

- EXAMINE:** a four bit condition code is set to indicate the contents of the top of stack (i.e., zero, positive, invalid, empty, etc.)
- DECOMPOSE:** the top of stack is decomposed into its exponent and significand and these two results are returned to the stack
- TEST:** the top of stack is compared to zero
- CONSTANTS:** a set of instructions that load internally stored constants onto the top of stack (i.e.,  $\pi$ , 0, 1, etc.)
- TAN:** takes the top of stack, Z as an argument, assuming that  $0 \leq Z \leq \pi/4$ , and returns two results, x and y such that  $y/x = \tan(Z)$ .
- ARCTAN:** takes the top two stack elements and returns the result Z such that  $Z = \arctan(y/x)$
- EXPONENTIAL:** takes the top of stack, x, assuming  $0 \leq x \leq 1/2$ , and returns  $2^x - 1$
- LOGARITHM:** takes the top two stack elements and returns  $y \cdot \log_2(x)$ .

In addition to the stack instructions listed above there are two instruction set optimizations. The first optimization is a set of arithmetic instructions that reference memory—one of the operands comes from the top of stack, the other from memory, and the result is returned to the top of stack. The operations which may use this optimization are ADD, SUBTRACT, SUBTRACT REVERSE, MULTIPLY, DIVIDE, DIVIDE REVERSE, COMPARE and COMPARE & POP. The four types of memory operands that can be referenced by these instructions are SHORT INTEGER (16 bits), INTEGER (32 bits), REAL (32 bits) and LONG REAL (64 bits). There are also STORE (without POP) instructions that reference the same four operand types. These instructions significantly reduce the number of instructions needed to evaluate a typical expression. For example, suppose R, X and Z are REAL, S and Y are LONG REAL, I is SHORT INTEGER and J and K are INTEGER. Then the expression

$$R := (S := (X/I + Y) / ((J - K) * Z))$$

is evaluated by the following code sequence:

Instruction	Memory Reference
LOAD REAL	X
DIVIDE SHORT INTEGER	I
ADD LONG REAL	Y
LOAD INTEGER	J
SUBTRACT INTEGER	K
MULTIPLY REAL	Z
DIVIDE REVERSE	
STORE LONG REAL	S
STORE & POP REAL	R

Without the additional memory referencing instructions the above expression would have required 14 instructions

and 3 stack elements instead of 9 instructions and 2 stack elements.

The second optimization involves internal stack addressing. There is a set of arithmetic instructions: ADD, SUBTRACT, SUBTRACT REVERSE, MULTIPLY, DIVIDE and DIVIDE REVERSE, that may take one operand from the top of stack (TOP) and the other operand from any stack element addressed relative to TOP (i.e.,  $TOP+i$ ,  $i=0,\dots,7$ ) and the result can be written over either operand. If the result is returned to the stack element (instead of the stack top) the instruction may either leave the top unaltered or pop the stack.

Thus the new instructions are:

(TOP) op (TOP+i)→(TOP)  
 (TOP) op (TOP+i)→(TOP+i)  
 (TOP) op (TOP+i)→(TOP+i) & POP

In addition to the arithmetic instructions mentioned, LOAD, STORE, STORE & POP, and EXCHANGE instructions may also refer to stack elements relative to TOP. For example LOAD TOP+i would load the contents of the  $i$ th stack element beneath the top onto the top of stack. These instructions allow stack elements to be used to accumulate results in loops and to hold common subexpressions. For example, suppose  $X(I)$  is an array of  $N$  REAL's and we want to calculate

$$R = \sum_{i=1}^N X_i, S = \sum_{i=1}^N i * X_i, T = \sum_{i=1}^N X_i^2$$

	INSTRUCTION	MEMORY REFERENCE
(R)	LOAD ZERO	
(S)	LOAD ZERO	
(T)	LOAD ZERO	
	LOAD REAL	X(I)
	ADD TOP+3	
	LOAD TOP+0 (this is the DUPLICATE TOP instruction)	
LOOP on I:	MULTIPLY TOP+0 (this is the SQUARE TOP instruction)	
	ADD & POP TOP+2	
	MULTIPLY SHORT INTEGER	I
	ADD & POP TOP+2	
	STORE & POP REAL	T
	STORE & POP REAL	S
	STORE & POP REAL	R

This stack addressing capability both minimizes memory referencing and permits loop accumulations to benefit from the extended range and precision of TEMPORARY REAL thus significantly attenuating the effect of roundoff error and making intermediate overflow or underflow practically impossible. Thus the 8087 may be thought of as a "pure" stack

machine with optimizations for memory and internal stack element addressing.

In addition to the computation instructions the 8087 has a set of administrative instructions for processor control and for status saving and restoring. In order to minimize context switching overhead there are single instructions, SAVE and RESTORE, that store and load respectively all 8087 volatile status. Also provided are instructions for loading and storing the 8087 status needed for software exception handling: exception flags and pointers to the offending instruction and datum. Finally, there is a 16 bit CONTROL WORD that may be loaded and stored. The contents of the control word dictate:

1. the rounding mode—there are four types of rounding.
2. the internal precision—results may be held internally in TEMPORARY REAL format but rounded to REAL (24 bit), LONG REAL (53 bit) or TEMPORARY REAL (64 bit) precision.
3. the mode of infinity arithmetic—there are two types of infinity closure, affine and projective, that will be explained later.
4. the response to exceptions—for each type of exception there is both a flag and an exception mask. According to the setting of the mask the 8087 either interrupts after setting the exception flag or it executes an on-chip microcoded exception handler and continues processing.

The usefulness of these controls and the power of the 8087 exception handling will be explained in the next section.

## USER BENEFITS

Many of the 8087 features confer significant user benefits. The benefits that are provided by five of these features will be described in this section:

1. the "extended" (TEMPORARY REAL) support format
2. the rounding modes
3. the on-chip exception handling
4. the modified stack architecture
5. the high performance.

One of the major innovations of the 8087 is the provision of an extended support format called TEMPORARY REAL. This format provides several significant advantages. Firstly, the 8087 should be thought of as having clean REAL (single) and LONG REAL (double) precision. By this we mean that not only is the arithmetic accurate but the commonly supplied system functions are also accurate to LONG REAL precision. For example if  $x$  is LONG REAL then  $e^x$ ,  $\ln(x)$ ,  $\tan(x)$ , etc., will all be accurate to within less than a unit in the last place of LONG REAL precision—in fact because of the on-chip primitive functions the logarithmic and trigonometric functions will be accurate to within a few units in the last place of TEMPORARY REAL precision. The

benefits of the TEMPREAL format can also be seen by examining its use in the most demanding function in the 8087's repertoire,  $X^y$ . In calculating this function one loses in extreme cases as many fraction bits in the answer as there are bits in the exponent of  $y$ ; if  $x$  and  $y$  are restricted to LONG REAL then  $z=x^y$  can lose about 11 bits in these extreme cases. This is a significant error in a function that is crucial for commercial calculations involving interest rates. By using TEMPORARY REAL and the 8087 logarithmic functions we can compute  $x^y$ , where  $x$  and  $y$  are LONG REAL, accurate to about a unit in the last place of LONG REAL precision. Besides providing accurate rate of return calculations we can also ensure that integral values of the arguments yield exactly what is expected (i.e.,  $2^3=8$  not  $8.00...01$ ).

Another benefit of the TEMPORARY REAL format is the ability to provide accurate libraries—mathematical, statistical, commercial, etc. The user of these libraries delivers his data in REAL or LONG REAL precision and receives his results in the same format. However, the library has used TEMPORARY REAL variables to perform internal calculations, thus protecting against not only roundoff errors but intermediate overflows and underflows (most over/underflows occur on intermediate calculations since usually the input and output lie within fairly narrow ranges). Most libraries make performance claims "in the absence of over/underflow." By judiciously using TEMPORARY REAL variables, libraries will often be able to ensure that the only over/underflows that occur either do not matter or are on output where they provide the user a necessary and useful warning result.

Another advantage of this support format is that code written by programmers who are unfamiliar with analyzing their programs for roundoff errors and other problems—this includes almost all of us—will much more often work correctly.

This is particularly true because of the extended stack—it is almost impossible and certainly inconvenient to compute on the 8087 without using the TEMPORARY REAL format. Consider for example the program discussed earlier for calculating the roots of a quadratic equation:

$$R_1:=(B+\sqrt{B^2-AC})/A$$

$$R_2:=(B-\sqrt{B^2-AC})/A$$

On a typical computer with no support format these formulas from high school math are subject to severe roundoff damage. However, because of the stack of TEMPORARY REAL registers, if the expressions are evaluated on the 8087, the support format is used automatically and invisibly for the sensitive parts of the calculation and the expressions are much more accurate. The 8087 stack thus makes "certified" software easier to write and makes it more likely that uncertified software is reliable.

A second major contribution of the 8087 to numerical computation is the capability of controlling the rounding mode. As described earlier there is a field in the CONTROL WORD of the 8087 that specifies how infinitely precise results are to be rounded to fit the designated format. If the correct result is exactly representable then that result is returned

regardless of the rounding mode. Otherwise the result can be specified to be any one of:

1. the nearest (if there are two then return the one with zero in the least significant bit—this avoids the usual bias)
  2. the next larger
  3. the next smaller
  4. the closer to zero (true truncation)
- (these modes are termed "directed rounding" (5))

Normally one would use the "nearer" rounding to compute the most accurate and statistically unbiased estimate of the correct result. Alternatively, by using the directed roundings, one can not only compute rigorous error bounds at crucial places in a program but also implement Interval Arithmetic (6,7). Interval Arithmetic, where operands and results are intervals instead of isolated numbers, completely encloses all rounding errors. Thus when a computation yields an interval result, the user knows that the exact result is contained in that interval. Interval Arithmetic can also be used to estimate the consequences of uncertainty in data. By entering the data as intervals enclosing any possible measurement errors, the width of the resulting intervals gives an indication of the sensitivity of the computation to those errors. Another use of Interval Arithmetic is to calculate, in a simulation, the effect on a system as a variable such as TEMPERATURE passes through a range of values. Professor W. Kahan of the University of California at Berkeley has written (8):

"No other feature would enhance safe numerical computation more than the provision of INTERVAL as a data type in FORTRAN as readily accessible as INTEGER or REAL."

If Interval Arithmetic is so useful why isn't it in widespread use? The main reason is that on a typical computer a rigorous Interval Arithmetic package can cost a factor of 100 to 300 over the ordinary floating-point arithmetic. On the 8087 this penalty is expected to be a factor of about 5. The implementation cost of providing the directed roundings was no greater than that of unbiased rounding so the value of the capability far exceeds its cost.

Another area where the 8087 makes significant contributions to safe but flexible software is exception detection and handling. Exception detection on the 8087 serves three main functions:

1. to report potentially fatal programming errors
2. to permit execution to be resumed after prearranged response to exceptional conditions
3. to allow functional extensions to the system.

Each type of exception detected by the 8087 has associated with it both a flag and a mask. (The exception masks are part of the CONTROL word and their value is set and saved by LOAD CONTROL and STORE CONTROL instructions.) When an exception occurs, the 8087 sets a flag and if the flag's mask is reset, an interrupt is generated. The interrupt procedure (exception handler) has access to the address of the instruction that caused the exception and the

address of the referenced datum (if any). If, on the other hand, the exception flag's mask is set, then the 8087 executes an on-chip microcoded exception handler that performs the second function described above: the instruction's response to the exception is "tailored" to that desired in the vast majority of cases. Execution resumes but the flag remains set until it is read and reset by software.

The exceptions that the 8087 detects and its response to them are explained below.

1. **INVALID OPERATION:** Stack overflow, stack underflow, indeterminate form ( $0/0$ ,  $\infty - \infty$ , etc.) or the use of a Non-Number (NaN) as an operand. An exponent value is reserved and any bit pattern with this value in the exponent field is termed a Non-Number and causes this exception.
  - a. Masked: If the exception was caused by using NaN's as operands then the NaN (the "larger" if both operands were NaN) is delivered as the result, otherwise a special NaN called INDEFINITE is returned.
  - b. Unmasked: Interrupt before any processing.

This exception is used for all of the purposes described. Indeterminate forms are usually fatal errors and should be reported—either immediately or by propagating INDEFINITE to the end of the program and thus discovering both the error and how it contaminates subsequent calculations. Stack over/underflow is also usually fatal but an ambitious exception handler could use this exception to extend the 8087 stack to memory. Finally, the NaN's can be used for both run time diagnostics and functional extensions. As an example of the former, one could fill uninitialized arrays with NaN's each of whose significands contains the value of its index. Thus a reference to an uninitialized array element would not only indicate that it was uninitialized but which one it was. An example of functional extension would be to use the NaN as a pointer into a heap of values that could not be stored in the specified format. This would make it possible to implement a nearly infinite exponent range.

2. **OVERFLOW:** The result is too large in magnitude to fit the specified format
  - a. Masked: Infinity with the sign of the correct result is returned.
  - b. Unmasked: An encoding of the true result is returned and then interrupt is signalled.
3. **ZERO DIVISOR:** The divisor is zero while the dividend is a finite non-zero number
  - a. Masked: Infinity is delivered with the sign as the XOR of the signs of the operands.
  - b. Unmasked: Interrupt before processing.

Both of these exceptions, if masked, generate infinities which are special bit patterns and must be dealt with in a safe, consistent manner by the 8087 in subsequent calculations. For this reason the 8087 recognizes infinities as valid operands and deals with them in one of two modes, AFFINE or PROJECTIVE, determined by a field in the CONTROL

WORD. The basic difference is that the affine treats all finite numbers as if  $-\infty \leq x \leq +\infty$  while in the projective mode  $\infty$  has no sign and cannot be compared to finite numbers. The affine mode is powerful but can give misleading results while the projective mode is always safe but not quite as useful as affine. The default is projective and this is the recommended mode unless a user has analyzed his program and is sure the affine mode is safe.

4. **UNDERFLOW:** The result is non-zero but too small in magnitude to fit in the specified format
  - a. Masked: The significand of the result is denormalized (shifted right) until the exponent is in range. This allows underflowed numbers "gradually" to become zero retaining as much information as possible and is called "gradual underflow."
  - b. Unmasked: An encoding of the correct result is delivered and then an interrupt is signalled.

Underflow is usually not a fatal error and by using gradual underflow (masking the exception) one can proceed, confident that the risk of undetected fatal underflow is commensurate with the risk of fatal roundoff damage (see 4).

5. **DENORMALIZED OPERAND:** At least one of the operands is denormalized, it has the smallest exponent but a non-zero significand.
  - a. Masked: The operation proceeds as if the operand were unnormalized.
  - b. Unmasked: Interrupt without processing. This exception is used to implement, via exception handlers, an optional mode of arithmetic described in the proposed IEEE Standard for Floating-Point Arithmetic (2) in which no unnormalized results are generated.
6. **INEXACT RESULT:** If the true result is not exactly representable in the specified format, the result is rounded according to the rounding mode, the flag is set and
  - a. Masked: Execution continues
  - b. Unmasked: Interrupt is signalled.

This exception is used to implement exact arithmetic in floating-point for, among other uses, accounting calculations and preconditioning (see 4).

Exception handlers are difficult to write, debug and maintain and they consume valuable memory space at run time. Therefore, we have provided, on the 8087, exception handling that will be ideal for the vast majority of situations. We recommend that most users mask all exceptions except INVALID OPERATION. With the built-in exception handling and reliable infinity arithmetic it is the only exception that is likely to be fatal. User exception handling software can thus be kept to a minimum.

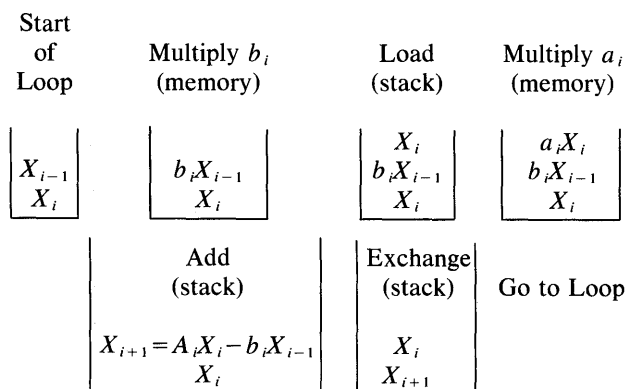
Another special feature of the 8087 to enhance performance and accuracy is the ability to select operands from and return results to internal stack elements. This stack element addressing mechanism which has already been described is useful for holding common subexpressions and for holding



accumulations during the execution of a loop. Another example will further illustrate its usefulness. An important calculation that is often found in the inner loop of numeric programs is the evaluation of recurrence relations. A particular example is the following three term recurrence

$$X_{i+1} = a_i X_i + b_i X_{i-1}, i = 2, \dots, N-1$$

Using a typical evaluation stack this computation would require, in addition to the add and two multiples, five memory references—four loads and one store. The evaluation on the 8087 stack requires only two memory references:



The "program" shown above illustrates a general principle. Almost all important numerical computations have inner loops that will benefit from the ability to access inner stack elements.

High performance was another of the important design goals of the 8087. It is difficult to compare 8087 performance with other machines since it is not feasible to obtain the same accuracy and reliability as the 8087 on even the largest mainframes. For example in executing a primitive instruction like MULTIPLY the 8087 provides:

1. A result with an extended precision and range
2. Correct unbiased rounding with optional direct roundings for error bounding
3. Reliable exception detection and safe, automatic handling
4. Forms of the instruction to minimize memory references.

No other computer—mainframe or minicomputer—integrates these features into a single architecture. But in addition to "architectural performance" a great deal of attention was given to raw instruction performance. For simplicity and execution speed the 8087 was implemented with an internal data path and ALU of 67 bits. There is a shifter that can shift left or right from 0 to 63 places in one clock cycle. This shifter was indispensable in normalization, data formatting and the transcendental functions which were evaluated using a modified CORDIC algorithm. The loops for MULTIPLY, DIVIDE and SQUARE ROOT were implemented with a hardware sequencer. MULTIPLY was optimized by checking for 40 least significant zeros and skipping

them in the multiply loop—this would occur if either operand were originally SHORT REAL or if either value were an integer and less than  $2^{25}$  in magnitude. The timing for several instructions demonstrates the 8087's performance.

Instruction	Execution Time (microseconds)
COMPARE	6
ADD (MAGNITUDE)	10
SUBTRACT (MAGNITUDE)	16
MULTIPLY	16,24*
DIVIDE	38
SQUARE ROOT	38

\* The shorter time applies if either operand were originally SHORT REAL as explained earlier.

Additional performance is gained by the overlapped execution of the 8086 (8088) and the 8087. The amount is hard to estimate but is definitely material.

## CONCLUSION

The architecture of the INTEL® 8087 has been described along with a review of its user benefits. The 8087 has unprecedented performance, reliability and capability—it can be used in any numerical application to provide a hundred-fold increase in mathematical performance over the 8086 or 8088 alone. In contributing to and being compatible with the proposed IEEE Floating-Point Standard the 8087 has carefully balanced safety with utility.

The many features of the 8087, when combined, can make it appear complex. Like a car's automatic transmission the 8087 is complex, but also like an automatic transmission the user need not see the complexity to reap the benefits of Interval Arithmetic, reliable rounding, safe automatic exception handling and an integrated support format that virtually eliminates intermediate over/underflows and makes intermediate roundoff error negligible. The 8087 removes many of the pitfalls of numeric computation.

## ACKNOWLEDGMENTS

I would like here to acknowledge some of the many people who contributed to the 8087. The architectural design was the joint work of Bruce Ravenal and myself, relying extensively on the advice of Professor W. Kahan of the University of California at Berkeley. Robert Koehler made significant contributions to the system aspects of the 8087 and Janis Baron designed the assembly language and implemented the 8087 Emulator—a software emulation for systems without an 8087. Rafi Nave and his engineering team in INTEL ISRAEL implemented the 8087—the largest microprocessor device yet in INTEL's history, and Dar-Sun Tsien carefully reviewed all aspects of the implementation. The management of INTEL must also be acknowledged for committing sig-

nificant resources to both implementation and promotion of a standard for reliable numeric data processing.

## REFERENCES

1. Palmer, J. (1977), "The INTEL Standard for Floating-Point Arithmetic," *Proc. COMPSAC*, 107-112.
2. Coonan, J., Kahan, W., Palmer, J., Pittman, T. and Stevenson, D. (1979), "A Proposed standard for Binary Floating-Point Arithmetic," *SIGNUM Newsletter*, October.
3. Coonan, J. (1980), "Specifications for a Proposed Standard for Floating-Point Arithmetic," *Computer*, January.
4. Kahan, W. and Palmer, J. (1979), "On a Proposed Floating-Point Standard," *SIGNUM Newsletter*, October.
5. Yohe, J. (1973), "Roundings in Floating-Point Arithmetic," *IEEE Trans. Computers*, Vol. C-22, No. 6, 577-586.
6. Moore, R. E. (1966), *Interval Analysis*, Englewood Cliffs, N.J.: Prentice-Hall.
7. Kahan, W. (1968), "A More Complete Interval Arithmetic," Lecture Notes for a course at University of Michigan, June 17-21.
8. Kahan, W. (1972), "A Survey of Error Analysis," *Information Processing 71*, North Holland Publishing Company, 1214-1239.



# Home computing—A vision in search of a marketplace: areas of needed research

by JOHN E. RUCHINSKAS, CHARLES W. STEINFELD and LYNNE L. SVENNING

Annenberg School of Communications  
Los Angeles, California

## INTRODUCTION

"The typical family of the late 1980's is a working couple with two children, one car, a small house and a rather well-structured way of living. . . . They have a home computer for paying bills, banking, monitoring their energy use, specialized research services and access to data for personal and business use."

So states Joseph Plummer, senior vice president of research at the Young and Rubicam advertising agency.<sup>1</sup> Like many soothsayers before him, Plummer assumes the home computer will be rapidly integrated into American households. Yet, a look at the predictions of a decade ago, which also promised the "magic" of home computers, calls into question just how inevitable the home computer really is.

The ubiquitous home computer has not emerged—and shows no new signs of doing so. Computing and memory costs are dropping, more and more individuals are being exposed to the computer at work each year and yet, the home computer bonanza remains a vision.

The natural question that comes to mind is, "Why hasn't the demand for home computers materialized?" Perhaps, the more fundamental questions are, "What *needs* do people have for computing functions in the home?" and "How can these needs be effectively addressed and marketed?"

By focusing on home computing *functions* rather than home computers, we are emphasizing the fact that any number of computer/telecommunication combinations can be used to accomplish the same task. Individuals might own or lease a terminal which is linked to a central computing facility, with little or no processing capability residing in the home. Alternately, one could own a stand-alone unit, as in the typical conception of the personal computer. Whatever the configuration, basic questions remain about people's willingness to purchase home-based computing functions, however they might be offered. We must move from wishful prognostication to systematic assessment of the *consumer's* (user's) point of view, and try to understand the factors that influence the mass-market appeal of home computing functions. We must look beyond the specialized hobbyists, computer programmers or adult toy enthusiasts who are cur-

rently purchasing computers or computing functions to the general consumer.

In order for computing to become a fixture in the American household, it must first and foremost offer the consumer clear benefits, either in terms of access to new services, convenience in fulfilling familiar tasks, or efficiency in managing one's daily life. Since any form of home computing represents a major capital investment for most consumers, it is unreasonable to expect widespread adoption, unless *perceived* needs of consumers are served.

This paper takes a consumer perspective in raising the kinds of issues that will shape the success or failure of future attempts to introduce home computing functions. Our goal is not the definition of a single research program, but rather to point out areas of uncertainty that cloud visions of the home computer future.

## SERVICE POSSIBILITIES

The first step in moving beyond today's specialized market lies in finding services that offer clear benefits to the general consumer. A full range have been proposed in the "blue-sky" literature. Those that appear most viable are briefly reviewed in the following paragraphs.

On the economic side of the ledger, there are home computer applications that promise the consumer greater convenience or cost savings in managing financial assets. Budget, task and financial analysis; electronic funds transfer, utility bill verification and exception reporting; energy management, home security and even investment portfolio management are all familiar possibilities. Greater convenience in record keeping (budget and bill management, tax reporting), automated accounting, increased security or even financial gains (energy conservation, portfolio management and financial accuracy) are proposed benefits. Business uses from the home offer the consumer both convenience and cost savings by eliminating trips to work.

A second set of activities that are easily amenable to computerization lie in the information domain. News, mail and classified advertising can all be computerized, targeted and sent electronically. Electronic libraries or filing cabinets con-

taining a range of information from personal documents to work-related data to recipes offer the consumer a more manageable information storage and retrieval option than current print-based options. Appointment or other scheduling information, as well as family messaging are frequently touted as possible uses. Computer assisted access to travel, entertainment and other directory information is an increasingly apparent reality, especially overseas.

Finally, in the area of education, programmed learning, word processing and student related coursework via computer offer seemingly attractive selling points.

Of course, entertainment is a function which cannot be overlooked. Home computers can offer electronic amusement and game functions, provide assistance in travel and entertainment reservations, and serve as a "creative companion" in music and graphic generation. Control of the entertainment/information environment with computer aid is also envisioned. Pre-scheduling, recording, controlling program or media access (especially for children) and playback are all realizable computer assisted options.

The oft-mentioned possibility of shopping from the home via electronic catalogues is the most "popular" of the interactive functions that promise the consumer convenience, time and energy savings. Other interactive alternatives that have also been proposed include opinion analysis/voting and nutrition/dietary analysis and management.

#### *Current consumer response to service possibilities*

While many consumer services can be provided through some combination of home terminal and computer capability, we have yet to assess whether the value assigned by the consumer exceeds the cost of providing the computer-assisted service. Consumer valuation will determine which services will be purchased by which consumers, in what combination, and for how much money.

Almost all projections of the potential demand for home computers, home terminals and interactive transaction services are based on the best guesstimates of "experts" as to what services are feasible and desirable.<sup>2,3</sup> The alternative to expert projections lies in extrapolating future demand from current sales figures.

Present estimates of home computer sales place the market at substantially less than half a million households during 1979. Lipoff<sup>4</sup> estimated there were 275,000 home computers sold through the end of 1978, while Nilles *et al.*<sup>5</sup> place personal computer sales at 250,000 for the same time frame. Both authors caution against using current owners as a model for future demand, since the present market is largely made up of hobbyists and individuals with previous computer experience. "The dominant portion of the growth in use of personal computers in the future depends upon the acceptance of the personal computer by quite a different set of individuals."<sup>6</sup>

As creatures of habit, we hold rather tenaciously to those that serve us well. The literature on change and innovation suggests that the rate of adoption of new products or new

ways of doing things is affected by:<sup>7</sup>

- the consumer's ability to observe the innovation in action;
- the consumer's ability to experiment or try the innovation without a great deal of risk;
- the perceived complexity of the innovation; that is, can the innovation be adopted relatively easily or is it difficult to understand and does it require a set of new skills before one can function effectively using the innovation;
- the compatibility of the innovation with the consumer's existing values, experiences and needs;
- and the perceived relative advantage the innovation has over the "old way."

Looking at attributes of home computers or even home computing functions with this diffusion of innovation perspective suggests a rather slow rate of adoption and reveals several potential barriers to general market acceptance.

- Consumers have relatively few opportunities to observe computers in action, especially in a home environment serving home needs. While the opportunities are steadily growing, the computing function is not always a readily visible aspect of the product offering. Therefore, the consumer may not observe and generalize about the value of home computing innovations.
- Home computers and computer assisted services are still rather expensive, thus a costly experiment for most consumers.
- While some computerized products/services require little from the consumer in the way of new knowledge or skills, the personal computer is another story. There is limited software available, which means the consumer must be satisfied with a few pre-programmed functions, or develop their own programs, which means learning new skills in many cases.
- Computers are generally improving their image in the "public eye." However, there is still a great deal of resistance to the notion of "computerized living." Computers are often viewed as the culprits in consumer's bad experiences with billing or banking. People also value the "human touch" and find in-person transactions often satisfy several needs, including the need for social interaction.
- Research suggests that consumers may have some difficulty seeing the relative advantage of computer assisted shopping, banking, budget-making or communication.

A series of studies completed by the Center for Communications Policy Research at the Annenberg School of Communications during the last seven years gives some indication of how the "average" consumer views the "desirability" of some services that might be introduced via home computers.

Respondents were introduced to the notion that some ac-

tivities they now engage in could be accomplished in the home with some combination of television, telephone and computers. They were then asked if they would be willing to purchase a variety of services for approximately the price of their basic monthly telephone bill. The services and the percentage of consumers willing to purchase these services are indicated in Table I. Consumers were most attracted to services involving entertainment, education, and civic functions. Banking, medical consultation, television visiting and accessing government information are only slightly less attractive, while notions of shopping and working from the home generally drew more negative responses. These figures should be considered with a grain of salt, since respondents may have been indicating a general positive or negative attitude toward the service rather than absolute willingness to purchase.

Consumer comments on why they would not be interested in such services most often reflected a need to get out of the house, the desirability of human contact in many transaction type activities, an inability to envision the service described, and perceived possibilities of abuses in security, privilege and privacy. They seem worried that computer and telecommunication assistance will depersonalize many activities, further concentrate activities in the home, un-

knowingly open their transaction and media behavior to scrutiny, and generally cause a deterioration in the quality of life they now enjoy.

Similar attitudinal barriers to home computing functions were noted in an investigation of potential users of electronic transaction services.<sup>9,10</sup> This study surveyed 325 Los Angeles residents about their attitudes toward electronic funds transfer, electronic shopping, and electronic mail systems, including willingness to use such systems. Respondents' attitudes about the reliability of computers proved to be the most influential variable in discriminating potential users from non-users. Those who felt "computers are less reliable than people" were least likely to say they would use any of the three systems. Concerns about privacy issues in the use of electronic services joined with demographic measures such as age and education in further discriminating individual willingness to use electronic services.

Overall, 24 percent said they were in favor of electronic mail, while 21 percent found electronic banking or shopping appealing. Significantly, more than a third of all respondents said they were neutral toward conducting at least one of these transactions electronically. Given the previously noted importance of attitudinal factors in influencing willingness to use these systems, it would seem especially critical to

TABLE I-Percent of Consumers Willing to Purchase Telecommunication/Computer Services\*

SERVICES	Sample Population						
	L.A. <sup>1</sup> 1973 (N=197)	San Diego <sup>2</sup> 1977 (N=500)	Sacra- mento <sup>2</sup> 1977 (N=400)	Fresno <sup>2</sup> 1977 (N=400)	Marin <sup>1</sup> 1978 (N=626)	Fort Worth <sup>1</sup> 1979 (N=200)	L.A. <sup>1</sup> 1979 (N=200)
EDUCATION-taking credit courses from the home via television	38.7	63.8	62.2	60.6	45.8	28	40.4
SHOPPING-using television to examine products, order by phone or interactive cable service	20.4	39.2	38.8	39.2	25.4	19	15.1
ENTERTAINMENT-first run movies and sports events	47.3	68.4	71.7	64.8	66.1	70	
MEDICAL CONSULTATION AND DIAGNOSIS-with computer assistance via a combination of telephone and television	29.2	45.2	40.4	48.4	26.5	24	30.0
BANKING-funds transfer, bill paying, etc. via telephone	13.2	56.4	58.6	60.3	49.8	37	49.3
CIVIC FUNCTIONS-such as voting, and driver's license renewal	27.4	73.1	74.2	72.8	62.3	57	68.2
VISITATION-via television with friends and relatives	38.1	64.3	53.6	50.1	47.0	50	47.9
ACCESSING GOVERNMENT INFORMATION-such as social security information or city council agenda	26.2	55.8	61.2	56.9	62.6	51	59.1
SELF IMPROVEMENT OR SKILL INSTRUCTION-such as carpentry or plant care	33.5	66.4	75.7	63.8	64.2	47	64.1
WORKING FROM THE HOME-with an electronic connection to the work place	16.1	38.8	33.1	31.9	27.5	28	28.8

\*Revised from Goldman<sup>8</sup>

<sup>1</sup>Random probability sample of adults, telephone interviews.

<sup>2</sup>Cluster sample of adult women, face-to-face interviews.

examine user attitudes and beliefs about computing functions if one were attempting to encourage the adoption of home-based computer services.

This brief discussion of our current state-of-knowledge about consumer attitudes and willingness to use and/or purchase telecommunication/computer "functions" indicates a general lack of information. We must know far more about the nature and extent of consumers' perceived needs, attitudes and intentions to purchase or use computing functions before we can assess market demand with any degree of accuracy and/or design products and services that will make the soothsayers' visions an evolving reality. Projecting short and long term consequences of consumer adoption is another important facet of missing information that may prove vital in allaying consumer fears and in designing a socially beneficial introduction of home computing.

**AN INFORMATIVE RESEARCH PROCESS**

Social science research can inform the process of creating reality out of visionaries' dreams of the future. The following pages outline a research process which reduces uncertainty about the marketplace, as well as the possible/probable consequences of widespread adoption of homing computing.

An assessment of consumer's attitudes, values and behavior before the introduction of a computer assisted lifestyle is at the core of the proposed research process. This *baseline information* provides product designers and marketers with insights about potential users and serves as a benchmark for measuring change. Market uncertainty can be reduced considerably through a program of *iterative formative evaluation*. Initial product design informed by baseline data is refined by a process of prototype development, analysis of consumer reaction, and product redesign.

In addition to helping shape viable market offerings, social science research can help anticipate the socio-economic consequences of widespread consumer adoption of new technologies. Anticipating direct and indirect impacts of consumer adoption and use of home computing products/services allows product designers and policy-makers to plan for both *socially* and *economically* beneficial outcomes. The formative evaluation process can be slightly expanded to develop information about the possible/probable impacts of new technologies on individual behavior and socio-economic institutions (Figure 1).

Short term changes in the attitudes, values and behaviors of consumers can be monitored in market-testing and early in the product diffusion process. These observed changes can be used to project larger socio-economic impacts.

Anticipatory product-design and policy formulation derived from consumer research can improve the likelihood that a desirable socio-economic future will emerge from the introduction of home computing. The following pages describe a research process that serves these ends.

*Formative evaluation*

A computer enthusiast can readily highlight the advantages/benefits accruing to the home user of computing prod-

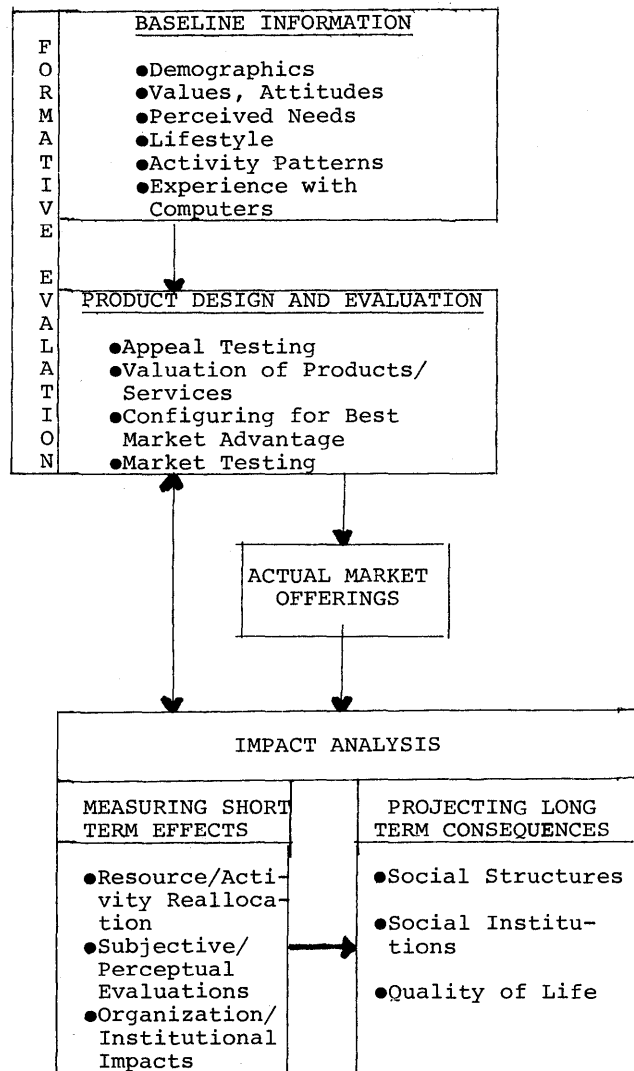


Figure 1-The research process.

ucts and services. The home user, on the other hand, often confronts the possibility of purchasing computing functions with a set of perceptions and attitudes that temper adoption and realization of such benefits. User factors must be researched and addressed before a consumer-oriented home computing marketplace will evolve.

An iterative *formative evaluation* process will yield practical information for designing computing innovations which have mass market viability.<sup>11</sup> As indicated, this process starts with gathering baseline information about consumers' values, lifestyles, activities, perceived problems, attitudes toward and experiences with computers, as well as products and services with computer components. The information can be used to design and refine product/service offerings in a manner which will optimize perceived consumer value. An accurately reflected market demand permits better "packaging" and marketing, which in turn promotes widespread adoption, and the evolution of a more user oriented home computing marketplace.

### Baseline information

Consumer *demographics, values, lifestyles, and perceived needs* are traditionally important factors accounting for variance in purchasing and utilization behaviors. The preliminary research previously cited indicates that age, education and socio-economic status are important factors influencing attitudes toward and willingness to purchase computing functions in the home. The research has been confined largely to California, and individual studies are biased in several other ways, such as all women populations, or an upper socio-economic survey area, or a totally Los Angeles population. While the results hold across most of the studies, there is a need for more in-depth research on more representative samples before we can generalize to the mass market.

Other factors may mediate the influence of demographic variables. We have yet to assess the relationships of *values* and other *lifestyle variables* to perceived need for, valuation of and willingness to purchase and/or use home computing functions. For example, the consumer's valuation of social interaction may be a better predictor of resistance to adopting home computing services than either age or socio-economic status. Or, consumers with an activity-rich lifestyle may see a value in saving time in this manner. Children in the home have also been shown to be an important lifestyle variable that influences the purchase of educational/information products and services. Some individuals prefer to separate their home and work lives, while others prefer an integration of life activities. Lifestyle preferences such as these are bound to influence the valuation of products and services which permit working from the home.

Very few of us are completely satisfied with our life experiences. Some of us fret over finances, others are bothered by crime in the streets. Some of us have difficulty finding time to spend with family members, others of us are bored and alienated. Identifying the common and even uncommon concerns of general consumers can provide "designers" and packagers with a set of perceived *needs* which can be addressed with communications and computer technology. An innovation that has been designed to meet *perceived consumer needs* has a head start in the race for consumer acceptance.

Consumer *experience* with computers and computing services is another type of useful baseline data. One's experiences and one's value system shape attitudes, which in turn have been shown to influence an individual's behavioral intentions.<sup>12</sup> There is a wide degree of variance in the general consumer's experience with computers, ranging from the computerized billings most of us receive, to daily interaction with computer terminals. These experiences crystallize positive or negative attitudes toward computing. Knowing what experiences characterize an individual's interaction with computing services and how that individual evaluates those experiences can provide important insights in designing products and services that will create generally positive experiences and enhance the probability of adoption.

A whole range of *attitudes* will influence individual inclinations to use computers or computing services. Attitudes

toward computers, concentrating activities in the home, depersonalizing transactions, energy conservation, privacy, security, television, etc. will all play a role in shaping the consumer's response to home computing. Identifying which attitudes are highly correlated with intentions to adopt or not adopt will give us a clearer picture of market characteristics.

For the most part, computer designers have produced products and services that appeal to an audience somewhat like themselves. Gathering more in-depth baseline information on a wider range of consumers is a valuable step in developing home computing products/services with a broader appeal. Baseline data gathered from a representative sampling of consumers should include not only the traditional measures cited earlier, but a cataloguing of daily activities and the values and satisfactions attached to these activities.

### Product design and evaluation

Formative evaluation is iterative in that the market is analyzed, products and services conceived, tested for appeal and valuation, designed and/or packaged, test marketed, refined and finally delivered to the general market. Throughout this process, empirical research can generate data which reveals how well products and services match market demands. The better the fit, the more likely the use or purchase.

Ideas or prototypes can be tested for *market appeal* using a variety of focus groups or survey techniques. *Early* consumer evaluation reduces the risk of sunk costs and market failure. Appeal is only one of the factors influencing consumer decisions. How the consumer *values* the product or service in the context of competing demands for time and financial resources is a crucial factor affecting innovation adoption. Can and will the consumer shift resources? What kinds of trade-offs can or will the consumer make in order to access or purchase computing services? How can computing products and services be designed to maximize the benefits and minimize the costs resulting from trade-offs made?

The *configuration* of product and service offerings also plays a major role in their market acceptance. For example, there appears to be a consumer orientation toward single purpose products. One has only to look to the typical kitchen to see specialized appliances, with purposes ranging from yogurt making to hot dog cooking. The home computing market seems headed in the same direction, with separate devices for controlling appliances, playing games, and word-processing. Is this a necessary step? Can the market be created in a more energy efficient, space efficient, resource efficient manner serving the "needs" of both producers and consumers of home computers? What factors will inhibit such developments?

Formative evaluation research can help answer these questions and in the process help create a home computing marketplace where product/service offerings fit consumers' needs and values. In-depth consumer research and product evaluation will generate the kind of information necessary to make home computing a reality.



### Impact analysis

Society is always altered by the introduction of technological innovations, sometimes in intentioned ways, often in unanticipated ways that prove less than desirable.<sup>13</sup> In recent years there has been a movement toward *inventing* or designing a desired future, rather than letting the future evolve in a happenstance manner. This requires feed-forward information about likely short term effects, as well as assessments of possible/probable long range impacts of technology adoption. Armed with systematically generated information regarding potential consequences, prescient planners and policy-makers in both the public and private sector can chart courses of action designed to avoid or mitigate undesirable consequences.

#### Measuring short term effects

Adopting home computer products/services will have significant immediate and continuing effects on the way individuals live their daily lives. Using computerized functions to substitute for, or augment, present behaviors will result in major *resource* and *activity reallocations*. The observable shifts in time, money and energy allocation may be accompanied by changes in individuals' *subjective/perceptual evaluation* of the particular activities in which they engage and their lives in general.

The most likely and noticeable effects of adopting home-based computing services will be changes in individual activity patterns. For example, computer assisted shopping from the home might result in less time spent shopping, less physical activity and gasoline consumption, and changes in individual spending habits. How will individuals allocate time formerly spent shopping? Will distinct patterns of activity reallocation emerge for different segments of the population? Will individuals reallocate travel savings into other out-of-home activities? Will individuals attempt to compensate for secondary characteristics of the shopping experience (e.g., social interaction) lost in computerized shopping? How?

Time allocation and activity pattern analysis before and after adopting computerized functions such as shopping will provide data on individual behavioral change. This analysis will indicate the types of direct effects and secondary impact that might be experienced with widespread adoption of computer assistance in the home.

Differences in the way individuals perceive or value activities are likely to accompany observable behavioral changes. For example, shopping may become a less valued activity due to the loss of social interaction. While time allocation and activity pattern analysis may reflect shifts in the valuation of life activities, they may not entirely capture the qualitative dimensions of the individual's experience. Asking individuals to assess their satisfaction with various life activities, as well as observing changes in the actual activity pattern, may provide feed-forward insights on the effects of home computing on perceived quality of life. Perceived valuation data may also yield data useful for

designing or modifying home computing products/services. For instance, should behavioral and valuation data indicate a strong attachment to social or group activity aspects of newly computer assisted functions, home-based electronic messaging and other services designed to enhance social interaction might be developed.

Attitudinal, valuational and behavioral individual effects are the most readily captured through existing research methodologies. Observation of change at the individual level can be used to assess certain short term *organizational* and *institutional* impacts. Aggregate shifts in resource allocation will mean some industries are likely to benefit from home computer services, while others may suffer, given maintenance of their current approach to the marketplace. For example, Ruchinskas<sup>14</sup> detected a decline in newspaper reading among consumers who had access to textual channels providing news and sports headlines via cable TV. Similarly, there was evidence that sports fans used more convenient text channels instead of TV news to gather sports information. Such cross impacts may follow the introduction of home computer products/services, unless existing service providers alter their offerings to provide unique consumer benefits.

To thoroughly assess short term effects at the organizational level, the proposed consumer-based analysis should be supplemented by monitoring other indicators. Specifically, this would involve tracking organizational response to observed shifts in consumer demands. This information will be vital in attempts to project long range institutional level impacts of widespread adoption of home computing.

#### Projecting long range impacts

Assessing and evaluating short-term effects creates the information base necessary for projecting long range social and economic consequences that may result from widespread adoption of new personal computing products/services. Impact domains of particular relevance with respect to home computing are *social structures*, *social institutions* and *quality of life*. Examples of possible changes in each of these domains are highlighted as a means of pointing to areas in which short term effects research might be used to project larger socio-economic consequences.

Our *societal structure* is in some ways reflected by individuals' access to information. Some have suggested the development of home information/educational computer-assisted services will exaggerate information or knowledge gaps. They envision a further exacerbation of "unequal information access," and a widening gap in information levels between the rich and the poor.<sup>15,16</sup> Examining the characteristics of purchasers, users and those interested in information related products and services, and assessment of pre-post knowledge levels, would provide a test of this hypothesis.

Other social structure ramifications may follow from possible reductions in interactions among members of different socio-economic "classes" due to the concentration of transactions in the home. This could reverse trends toward in-

tegration and cause a loosening of the social fabric. Activity analysis could reveal the beginnings of changes that might have long term implications for changing social patterns.

Services offered through home computers and/or interactive terminals are likely to have impacts on existing *social institutions*. Widespread adoption of computer assisted instruction in the home would definitely impact our educational institutions and alter the manner in which educational services are delivered to the public. Transportation institutions could experience major impacts from a reallocation of the work force, due to increased working from the home or local work centers via computer linkages. Potential socio-economic impacts in this area have been discussed in the telecommunication literature.<sup>17</sup> Examples of possible institutional impacts are myriad, but research assessing the nature and/or likelihood of these impacts is almost nil.

Finally, all of the possible and probable changes we have discussed interact to influence the quality of life (QOL). Quality of life is a function of both the "objective" environment (physical, social, economic and political) and subjective perceptions, attitudes and experiences of individuals living in that environmental context.<sup>18</sup>

Assessments of life and activity satisfaction will provide some indication of the impacts home computing has on individuals' perceived quality of life. In part, product design and planning represents a further effort to maintain a high quality of life by designing products and services that reflect consumer needs and values. However, careful monitoring of changes in organizational and institutional structures is also necessary. This is evidenced in past QOL research, which found the family institution to be the strongest factor influencing individuals' perceived quality of life.<sup>19</sup>

### Policy implications

It must be recognized that information provided by projections of long range socio-economic consequences lies in the domain of public policy making as well as private industry. Research results identifying potential tendencies toward increasing "information gaps" may highlight policy issues which ultimately require government intervention to facilitate "equal" access to educational and information services. Policies may also take the form of legislative restraints should negative social consequences seem likely based on the empirically-based projections. Private industry must therefore anticipate socio-economic consequences and subsequent government action which would alter the shape of the marketplace.

### CONCLUSION

Research can reduce the uncertainty surrounding both the development of the home computing marketplace and the impacts of home computing on the quality of our lives. This research must begin with a more careful study of potential users and proceed to examine human behavior in home settings to which home computing has been introduced. After

determining how human behavior changes with access to and utilization of home computing, researchers can begin to more accurately assess the possible/probable socio-economic consequences of a computer assisted lifestyle.

We have described a *research process* designed to maximize both the short and long term benefits of introducing home computing functions. The formative evaluation data that is essential to successful introduction of computing functions also provides us with baseline measures for judging future changes. Throughout the product design and market testing phases, the immediate effects of using home computing functions can be monitored. With this information, we can project what the impact would be if these observed changes took place on a large scale. This information feeds back to decision makers to shape the next round of product design and market introduction. In-putting "impact" data early in the product development cycle, before parties have a large vested interest or capital investment in existent technology minimizes the risks of undesirable consequences.

Such a concerted research effort represents an attempt to maximize both the *economic* and *social* benefits of home computing. In an era where societal resources are becoming recognized as a finite and valuable commodity, we cannot miss this opportunity to direct technological change in a manner that maximizes its utility for all stakeholders.

### REFERENCES

1. *Advertising Age*, November 5, 1979.
2. Baran, Paul (1971), "Potential Market Demand for Two-way Information Services to the Home," 1970-80, Institute for the Future, Palo Alto.
3. Bortz, Paul (1973), "Technological Innovations in Video and Their Potential Market," Denver Research Institute, NTIS Pub. Com-73-1145.
4. Lipoff, Stuart (1979), "Mass Market Potential for Home Terminals," *IEEE Transactions on Consumer Electronics*, CE-25(2): 169-184.
5. Nilles, J. et al. (1979), "Personal Computer, Technology, Users and Uses," Report No. 01P/PCTA-79-1, University of Southern California, Office of Interdisciplinary Affairs, February 1979.
6. *Ibid.*, pp. 3-18.
7. Rogers, Everett and Floyd Shoemaker (1971), *Communication of Innovations*, New York, The Free Press.
8. Goldman, Ronald (1979), "Demand for Telecommunication Services in the Home," paper presented at the ICA Convention, Philadelphia, Pa., May.
9. Schwalbe, Ted (1977), "Potential Adopters of EFT and Electronic Mail Systems," Center for Communications Policy Research, Annenberg School of Communications at the University of Southern California, Los Angeles.
10. Schwalbe, Ted (1978), "Diffusion of Electronic Transactions," Center for Communications Policy Research, Annenberg School of Communications at the University of Southern California, Los Angeles.
11. Martin, Thomas (1978), "Can Interaction With Computers Be Made Enjoyable For the Public," *Proceedings of the International Conference on Cybernetics and Society*, Vol. II., Tokyo-kyoto, Japan, November 3-7.
12. Fishbein, M. and Ajzen I. (1975), *Belief, Attitude, Intention and Behavior: An Introduction to Theory and Research*, Reading, Mass, Addison-Wesley.
13. Ellul, Jacques (1964), *The Technological Society*, (trans. by John Wilkinson), New York, Alfred A Knopf.
14. Ruchinskas, John (1979), "The Consumer in the Electronic Marketplace," Center for Communications Policy Research, Annenberg School of Communication, Media '90 Report #2-79, Los Angeles.
15. Tichenor, et al. (1970), "Mass Media and Differential Growth in Knowledge," *Public Opinion Quarterly*, 34:158-170.

16. Katzman, N. (1974), "The Impact of Communication Technology: Promises and Prospects," *Journal of Communication*, 24:47-58.
17. Nilles, J. et al. (1976), *The Telecommunications-Transportation Tradeoff: Options for Tomorrow*, New York, John Wiley.
18. Hornback, K. and Shaw, R. (1973), "Toward a Quantitative Measure of the Quality of Life," in *EPA-The Quality of Life Concepts*, Washington, D.C., The Government Printing Office.
19. Andrews, F. and Withey, S. (1974), "Developing Measures of Perceived Life Quality: Results from Several National Surveys," *Social Indicators Research*, 1:1-26.

# 1980 NATIONAL COMPUTER CONFERENCE COMMITTEES

## PROGRAM COMMITTEE

### *Chairman*

Don B. Medley  
Moorpark College  
Moorpark, CA

Eric D. Carlson  
IBM Corporation  
San Jose, CA

Don Reifer  
Software Management Consultants  
Torrance, CA

### *Secretary*

Karen Nelson  
Moorpark College  
Moorpark, CA

Lance A. Leventhal  
Emulative Systems  
San Diego, CA

Erica M. Rounds  
Technology Service Corporation  
Santa Monica, CA

### *Vice Chairman*

Linda Taylor  
SDC, Manager Technical Audits  
Santa Monica, CA

G. "Jack" Lipovski  
Electrical Engineering Dept.,  
University of Texas  
Austin, TX

James R. Stallard  
General Dynamics  
St. Louis, MO

### *Committee Advisor*

Sakti Ghosh  
IBM Corporation  
San Jose, CA

Harvey Marks  
Informatics Inc.  
Canoga Park, CA

Todd P. Ziesing  
Bank of California  
San Francisco, CA

### *Technical Area Coordinators*

John C. Biddle  
California State University  
Bakersfield, CA

Susan H. Nycum  
Chickering and Gregory  
San Francisco, CA

Mark Spitz  
Informatics Inc.  
Canoga Park, CA

## CONFERENCE STEERING COMMITTEE

### *Conference Chairman*

Herbert B. Safford  
GTE Data Services, Inc.  
Marina Del Ray, CA

### *Special Activities Chairman*

Dorothy M. Parson  
Torrance, CA

### *Operations Chairman*

Mary Rich  
Imperial Computer Services, Inc.  
Torrance, CA

### *Vice Chairman*

Guy H. Dobbs  
Xerox Electro-optical Systems  
Pasadena, CA

### *Personal Computing Co-chairman*

Lewis A. Whitaker  
Innovative Computer Products  
Tarzana, CA

### *Communications Chairman*

Ted E. Lorber  
C. Itoh Electronics  
Los Angeles, CA

### *Program Chairman*

Don B. Medley  
Moorpark College  
Moorpark, CA

### *Personal Computing Co-chairman*

Lawrence Press  
Small Systems Group  
Santa Monica, CA

### *Registration Chairman*

Ronald W. Colman  
California State University  
Fullerton, CA

### *Professional Development Chairman*

James H. Weiss  
GTE Data Services, Inc.  
Tampa, FL

### *Finance Chairman*

Richard B. Blue, Sr.  
TRW Defense and Space Systems  
Group  
Redondo Beach, CA

### *Human Resources Chairman*

Thomas Ilas  
San Diego Gas & Electric  
San Diego, CA

*NCC Committee Liaison*

Jerry Koory  
On-Line Business Systems Inc.  
San Francisco, CA

*Pioneer Day Chairman*

Mort Bernstein  
System Development Corp.  
Santa Monica, CA

*Advisor*

Robert W. Rector  
CMAC  
Cranston, RI

*AFIPS Liaison*

Jerry Chiffriller  
AFIPS Headquarters  
Arlington, VA

## NCC '80 AREA DIRECTORS

Russell Abbott  
California State University  
Northridge, CA

Barry W. Boehm  
TRW  
Redondo Beach, CA

Eric D. Carlson  
IBM Corporation  
San Jose, CA

Wesley W. Chu  
UCLA-Westwood  
Los Angeles, CA

Jim Carlisle  
Office of the Future, Inc.  
Guttenberg, NJ

Lorraine M. Duvall  
IIT Research Institute  
Rome, NJ

Roger Firestone  
Sperry Univac  
Blue Bell, PA

Kurt F. Fischer  
Computer Sciences Corporation  
Falls Church, VA

Herbert Hecht  
Sohar Inc.  
Los Angeles, CA

Alyce Jackson  
TRW  
Redondo Beach, CA

Susan Landa  
Bunker Ramo  
Westlake Village, CA

Lance A. Leventhal  
Emulative Systems Co.  
San Diego, CA

Bennet P. Lientz  
UCLA  
Los Angeles, CA

G. Jack Lipovski  
University of Texas  
Austin, TX

Vincent Lum  
IBM Research Lab  
San Jose, CA

Harvey Marks  
Informatics, Inc.  
Canoga Park, CA

Ephraim R. McLean  
University of California  
Los Angeles, CA

Peter Neuman  
SRI International  
Menlo Park, CA

Susan Nycum  
Chickering and Gregory  
San Francisco, CA

Jon Prescott  
J&S Associates  
Fremont, CA

Larry Putnam  
QSM Inc.  
McLean, VA

Don Reifer  
Software Management Consultants  
Torrance, CA

Leigh S. Rosenberg  
Jet Propulsion Lab  
Pasadena, CA

Erica M. Rounds  
Technology Service Corp.  
Santa Monica, CA

Eugene Smith  
USDA, Comm. and Data Services  
Division  
Beltsville, MD

Roland Spaniol  
Eastern Illinois University  
Charleston, IL

James Stallard  
General Dynamics  
St. Louis, MO

T. Straeter  
General Dynamics  
St. Louis, MO

Linda Taylor  
SDC  
Santa Monica, CA

Andrew Tescher  
Aerospace Corp.  
Los Angeles, CA

Ken Thurber  
Sperry Univac  
St. Paul, MN

Frank T. Tung  
IBM Research Lab  
San Jose, CA

Rein Turn  
California State University  
Northridge, CA

Walter E. Ulrich  
Houston, TX

Fred Weingarten  
Office of Technical Assessment  
Washington, DC

Raymond Yeh  
University of Maryland  
College Park, MD

Todd Ziesing  
Bank of California  
San Francisco, CA

## NCC '80 SESSION CHAIRMEN

Russell Abbott  
California State University  
Northridge, CA

Don Aharonian  
DEC  
Weston, MA

Helen M. Alex  
USGS, Conservation Division  
Washington, DC

Boyd Alexander  
U.S. House of Representatives  
Washington, DC

Roger Allen  
Computer Sciences Corporation  
El Segundo, CA

Bharat Bhargava  
University of Pittsburgh  
Pittsburgh, PA

Barry W. Boehm  
TRW  
Redondo Beach, CA

John Brackett  
Softtech Microsystems  
San Diego, CA

Edgar Bristol  
The Foxboro Company  
Foxboro, MA

Louis J. Brocato  
USDA, SEA, CDS  
Beltsville, MD

A. Windsor Brown  
General Automation  
Anaheim, CA

Thomas C. Brown  
Computer Sciences Corporation  
Silver Spring, MD

Robert Campbell  
Advanced Information Management,  
Inc.  
Woodbridge, VA

Jim Carlisle  
Office of the Future, Inc.  
Guttenberg, NJ

Coleen Cayton  
Denver Public Library  
Denver, CO

Ned Chapin  
Infosci, Inc.  
Menlo Park, CA

Peter Chen  
University of California  
Los Angeles, CA

Ira W. Cotton  
National Bureau of Standards  
Washington, DC

Daniel J. Couger  
University of Colorado  
Colorado Springs, CO

C. Paul Davis  
Innovative Computer Products  
Tarzana, CA

Carl G. Davis  
Data Processing Directorate  
Huntsville, AL

William R. Deitrick  
Mini-Micro Systems, Inc.  
Anaheim, CA

Ed Dodson  
GRC  
Santa Barbara, CA

Larry Druffel  
DARPA/IPTO  
Arlington, VA

Lorraine M. Duvall  
IIT Research Institute  
Rome, NY

William E. Farley  
USDA  
Beltsville, MD

Roger Firestone  
Sperry Univac  
Blue Bell, PA

Kurt F. Fischer  
Computer Sciences Corporation  
Falls Church, VA

Werner Frei  
USC Medical Imaging Group  
Marina Del Ray, CA

Leonard B. Gardner  
San Diego, CA

Edward L. Glaser  
Ampex Corporation  
El Segundo, CA

F. E. Graham  
Computer Devices, Inc.  
Birmingham, MA

Paul Gray  
Cox School of Business, SMU  
Dallas, TX

Hal Hart  
TRW  
Redondo Beach, CA

Henry Heffernan  
Washington, DC

Jeffrey Hoffer  
School of Management  
Cleveland, OH

Lance J. Hoffman  
George Washington University  
Washington, DC

Bob Holland  
Database Design Corp.  
Ann Arbor, MI

David L. Holzman  
University of Southern California  
Los Angeles, CA

Joseph E. Izzo  
Joseph Izzo Associates  
Chicago, IL

Alyce Jackson  
TRW  
Redondo Beach, CA

Bob Johansen  
Institute for the Future  
Menlo Park, CA

Steven Kartashev  
University of Nebraska  
Lincoln, NEB

Svetlana Kartashev  
University of Nebraska  
Lincoln, NEB

Steve Kimbleton  
National Bureau of Standards  
Washington, DC

Frank Kline  
International Data Corp.  
Waltham, MA

Rob Kling  
University of California  
Irvine, CA

Benn R. Konsynski  
University of Arizona  
Tucson, AZ

Neal Koss  
Society for Computer Medicine  
Torrance, CA

Philip Kraft  
State University of New York  
Binghamton, NY

Edward Y. S. Lee  
TRW  
Redondo Beach, CA

Stephen Levine  
Lawrence Livermore Labs.  
Livermore, CA

David Lien  
CompuSoft, Inc.  
San Diego, CA

Bennet P. Lientz  
University of California  
Los Angeles, CA

Myron Lipow  
TRW  
Redondo Beach, CA

Mary E. S. Loomis  
University of Arizona  
Tucson, AZ

Eugene Lowenthal  
Intel Corp.  
Austin, TX

Ephraim R. McLean  
University of California  
Los Angeles, CA

Dennis McLeod  
University of Southern California  
Los Angeles, CA

Richard E. Merwin  
George Washington University  
Washington, DC

Edward F. Miller  
Software Research Associates  
San Francisco, CA

John Mitchell  
Georgia Institute of Technology  
Atlanta, GA

John Musa  
Bell Laboratories  
Whippany, NJ

Jack Nilles  
University of Southern California  
Los Angeles, CA

David L. Peters  
Racal-Vadic, Inc.  
Sunnyvale, CA

Lori Pitchell  
Burroughs Corp.  
Atlanta, GA

William K. Pratt  
Compression Laboratories, Inc.  
Copertino, CA

Jon Prescott  
J&S Associates  
Fremont, CA

Kenneth W. Rind  
Xerox Development Corporation  
Stamford, CT

David C. Rine  
Western Illinois University  
Macomb, IL

Leigh S. Rosenberg  
Jet Propulsion Lab  
Pasadena, CA

Alexander D. Roth  
AFIPS Washington Office  
Arlington, VA

Donna Shepherd Rund  
Pacific Telephone  
Martinez, CA

Sabina Saib  
GRC  
Santa Barbara, CA

Norman Schneidewind  
Naval Postgraduate School  
Monterey, CA

Nan Shu  
IBM Research  
San Jose, CA

Eugene Smith  
USDA, Comm. and Data Services  
Division  
Beltsville, MD

David Snyder  
Walt Disney Productions  
Glendale, CA

Norm Sondheimer  
Sperry Univac  
Blue Bell, PA

Roland Spaniol  
Eastern Illinois University  
Charleston, IL

James Stallard  
General Dynamics  
St. Louis, MO

Leon Stucki  
BCS  
Seattle, WA

Stanley Su  
University of Florida  
Gainesville, FL

John Swearingen  
U.S. Senate  
Washington, DC

Richard H. Thayer  
Sacramento Air Logistics Center  
McClellan Air Force Base, CA



Douglas J. Theis  
The Aerospace Corporation  
Los Angeles, CA

Charles Tucker  
Twentieth Century-Fox  
Beverly Hills, CA

Rein Turn  
California State University  
Northridge, CA

Walter E. Ulrich  
Houston, TX

Charles R. Vick  
BMDATC  
Huntsville, AL

Raymond P. Voith  
Motorola, Inc.  
Austin, TX

Gerald R. Wagner  
Execucom, Inc.  
Austin, TX

Stephen T. Walker  
Dept. of Defense  
Washington, DC

Virginia Walker  
Dept. of Energy  
Arlington, VA

William Ward  
Theater Arts Department  
Los Angeles, CA

Caroline M. Watteuw  
Office of the Future, Inc.  
Guttenberg, NJ

Claude Wiatrowski  
University of Colorado  
Colorado Springs, CO

Tom Wiener  
DARPA  
Arlington, VA

Raymond Yeh  
University of Maryland  
College Park, MD

---

## NCC '80 REFEREES

On behalf of the 1980 Program Committee, we wish to thank the following referees for their efforts. Although an attempt was made to list all referees, there are undoubtedly some omissions, for which we apologize.

D. B. MEDLEY, EdD, CDP  
Editor

Abbey, Duane  
Abbott, Russell  
Adams, Russell  
Agrawal, Dharma  
Agrawala, Ashotz  
Ahuja, Pratap  
Ahuja, Sanjiv  
Ahuja, Vijay  
Aiken, Robert  
Al-Fedaghi, Sabah  
Amenta, Joyce  
Ames, Stanley  
Andrison, John  
Antal, J. R.  
Archibald, Julius  
Arps, Ron  
Aylor, James

Baer, J-L  
Bai, William G.  
Baird, George  
Baker, F. T.  
Barnes, Bruce  
Bateman, Barry L.  
Bauer, Michael  
Baxter, Brent  
Belford, Geneva  
Bering, Doug  
Berk, Toby  
Bhargava, Bharat  
Bise, Robert  
Blomgren, George  
Bork, Alfred  
Borko, Harold  
Bracey, R. D.  
Brocato, Louis  
Brown, Nander  
Brown, Russell K.  
Bruell, Steven  
Burton, William

Campbell, Roy  
Cannon, George  
Capparo, Gerrard  
Carey, Bernard  
Carroll, B. D.  
Carter, W. C.  
Castillo, Pedro  
Chapin, Ned  
Charney, R. B.

Charp, S.  
Cheydleur, B. F.  
Chow, Yuan-Chien  
Cieslowski, Richard  
Clema, Joe  
Clemons, Eric  
Cobb, Gary  
Coke, Esther  
Cook, Betty Jane  
Coopriider, Lee  
Couperus, J.  
Cowan, George  
Crenshaw, Edsel

Daniels, Walter  
Danielsson, Per-Erik  
Danner, Lee  
Davida, George  
Davis, Alan  
Day, William  
de Jong, Kenneth  
Deb, Ashoke  
Dekock, Arlan  
Denenberg, Stewart A.  
Dewdney, A. K.  
Dittrich, Klaus  
Dixon, Louis  
Druseikis, Frederick  
Dunaway, Donna  
Dunbrow, Ardyn E.  
Duncan, Karen  
Dutton, Ronald  
Dwyer, S. J.

Eastman, Caroline  
Eccles, William  
Egen, John  
Ellis, Clarence A.  
Emery, James  
Ernst, Ronald L.  
Erwin, Harry  
Estrin, Thelma

Fay, Tim  
Feldman, Michael B.  
Finfer, Marcia  
Firestone, Roger M.  
Flinchbaugh, B. E.  
Flynn, Robert  
Fong, Elizabeth

Freeman, Harvey  
Friedman, Lee A.  
Fu, K. S.  
  
Galkowski, Jan  
Gannon, Thomas F.  
Garcia, Oscar N.  
Gehani, Narain  
Geraghty, John  
Giordano, J. V., Jr.  
Goel, Amrit  
Goldhirsh, Isadore  
Goldman, Neil  
Gonzales, Mario J., Jr.  
Gottlieb, Allan  
Granlund, G.  
Grayson, Alice L.  
Green, Teresa  
Greenleaf, James F.  
Grosch, Audrey  
Gross, Arthur G.  
Gupta, Ram

Hakozaki, Katsuya  
Hall, Ernest L.  
Hamblen, John  
Hanna, William  
Hart, P. E.  
Hartson, Rex  
Harvey, Thomas  
Hattori, Mitsuhiro  
Haynes, Gregory A.  
Hecht, Herbert  
Hein-Gal, Moshe  
Heller, Paul S.  
Herman, G. T.  
Hoffman, Lance  
Hollaar, Lee  
Holmes, Harvard  
Hoover, L. Ronald  
Hopper, Grace  
Horsted, Burt  
Hoyt, Patrick  
Hua, Cecil T.  
Huang, T. S.

Ichikawa, Tadao  
Idesawa, Masanori

Jacobs, Barry

Jensen, Douglas  
Jette, Christina  
Johnson, L. Arnold  
Johnson, Ark  
Jordan, Harry F.

Kahn, Kevin  
Kamal, Samir  
Kampen, Garry  
Kaufman, Arie  
Kooiman, Donald  
Koory, Jerry L.  
Kornfield, N. R.  
Kubitz, W. J.  
Kunz, Gregory A.

Landis, Carolyn P.  
Lee, Mary Jane  
Lee, Theodore  
Leinbaugh, Dennis  
Levine, M. D.  
Lien, David  
Little, Elizabeth R.  
Little, Joyce  
Lockett, Joann  
Long, Harvey  
Lucido, Anthony P.

Machover, Carl  
Madrigan, Orlando  
Madron, Beverly  
Maekawa, Mamoru  
Magel, Kenneth  
Magnuson, Waldo  
Maher, Austin  
Mallett, Patrick W.  
Mander, K. C.  
Maniotes, John  
Maskewitz, Betty  
Mathews, Walter  
Matsushita, Yutaka  
McAllister, David  
McCrea, Donald  
McDonald, Nancy  
McHenry, J. A.  
McHenry, Stephen  
McLeod, Dennis  
McMahon, Edith  
McNulty, Lynn  
Metzner, John  
Miller, Charles  
Miller, Dale  
Miller, Mark Leslie  
Mittal, Sanjay  
Modesitt, Kenneth  
Murphy, Robert

Naemura, Kenji  
Nash, Michael

Navlakha, Jainendra  
Nelson, Victor  
Nestman, Chadwick  
Netravali, A. N.  
Neumann, Peter  
Nielsen, Norman  
Nilsson, Arne A.  
Nutt, Gary

O'Kane, Kevin  
O'Neal, Beverly  
Oliver, Ellen

Peacock, Walter L.  
Peralta, L. A.  
Perry, J. M.  
Pfaltz, John L.  
Pottinger, Hardy  
Powell, John  
Pradhan, Dhiraj K.  
Preston, K.  
Prewitt, Judith  
Profio, E.

Rajaler, Sarah A.  
Raskin, Jef  
Reddy, Raj  
Reho, Andrew M.  
Rein, Robert  
Resta, Edward V.  
Richardson, Debra J.  
Riddle, William  
Robb, Richard A.  
Roberts, Eric  
Rodman, Robert  
Rosenbaum, Susan L.  
Rosin, R. F.  
Roth, R. Waldo  
Rowa, Per  
Rulifson, J. F.  
Ruschitzka, Manfred

Sangal, Rajeev  
Sashin, D.  
Savage, Carla  
Schafer, David  
Scheuermann, Peter  
Schneider, G. Michael  
Schultz, David J.  
Segal, Ronald  
Shaffron, Nancy  
Shapiro, Michael  
Shetler, A. C.  
Shoquist, Marc  
Siegel, H. J.  
Simmons, Dick B.  
Sitkin, Irwin J.  
Skeel, Robert

Sklansky, J.  
Smid, Miles  
Smith, Eugene  
Smith, Robert  
Smoot, O. R.  
Snyder, Wesley E.  
Soma, Takashi  
Sondheimer, Norman  
Spaniol, Roland D.  
Stavely, Allan  
Steele, Stuart  
Stevens, D. F.  
Stevens, W. Richard  
Stuck, B. W.  
Sudo, Masaru  
Svigals, Jerone  
Szolovits, Peter

Tai, K. C.  
Tausner, Miriam  
Taylor, Linda T.  
Taylor, Robert  
Teichroew, Daniel  
Thurber, K. J.  
Tinaziepe, Cihan  
Tomaru, Keisuke  
Towsen, James  
Tucker, Edwin K.  
Turn, Rein

Updegrove, Daniel A.  
Usanis, Richard A.

van Cleemput, W.  
van Name, Mark L.  
Vandergaag, Michael  
Varshney, Pramod K.  
Verba, John

Warren, Jim  
Wasserman, Reuben  
Waterman, David  
Weiss, Stephen  
Wesselkamper, T. C.  
Whiting-O'Keefe, Patricia  
Wiederhold, Gio  
Wileden, Jack  
Williams, John  
Wolfson, Seymour J.  
Wong, Tom  
Worrest, Ralph W.  
Wu, Chaun-Lin  
Wynne, James A.

Yamamoto, Masahiro  
Yao, Bing  
Yasnoff, William  
Yovits, Marshall

## NCC '80 SPEAKERS AND PANELISTS

Ahlgren, Dave  
 Al-Shaikh, Al  
 Albright, Thomas G.  
 Alexander, Boyd  
 Amamiya, Makoto  
 Anderson, Howard  
 Anderson, Mary Pastel  
 Anderson, Ronald E.  
 Antal, Joe  
 Appleton, Daniel  
 Arnold, Robert  
 Audolph, Jim

Baird, George N.  
 Ball, Marion J.  
 Balzer, Robert  
 Bamburg, Ronald  
 Barlow, Mel R.  
 Barrow, N. G.  
 Bateman, Joan P.  
 Baumann, L. S.  
 Baxter, Brent  
 Beamer, Gary D.  
 Belady, Lazlo  
 Belair, Robert  
 Belz, Frank C.  
 Berlinger, Eli  
 Bernstein, Lionel M.  
 Bethancourt, Michael  
 Blasgen, Michael  
 Bloom, Mitchel  
 Bolles, R. C.  
 Borden, Chester S.  
 Bowen, John  
 Bracker, William  
 Brand, Donald A.  
 Braun, Christine L.  
 Briggs, Warren  
 Brogie, Roger  
 Brown, David G.  
 Brown, Thomas C.  
 Busenberg, Stavros  
 Buxton, John N.

Candlin, Jim  
 Capitant, Patrice  
 Carlson, William E.  
 Cartman, Diane  
 Catmul, Edward  
 Chang, Hsu  
 Charlu, D. P. S.  
 Cheatham, Tom  
 Chenub, David  
 Clark, Lori  
 Cochran, Jack  
 Codd, E. F.

Colton, Kent W.  
 Comer, Douglas  
 Conner, Guy  
 Cook, Carolyn L.  
 Cooley, Michael  
 Copeland, George  
 Costa, John  
 Cragon, Harvey  
 Crane, Steven N.  
 Cross, Thomas  
 Crowley, Jeri

Daniels, Gary  
 Davis, Alan M.  
 Davis, Joe T.  
 Davis, Richard  
 Dawson, Kenneth  
 Dawson, Stephen  
 De Maio, Harry B.  
 De Millo, Richard A.  
 de Valpine, Jean E.  
 Deitrick, William R.  
 Derico, Cathy  
 Doll, Dixon  
 Donovan, Thomas  
 Doshay, Irving  
 Doyle, Richard F.  
 Draper, William  
 Druffel, Larry E.  
 Dubrowski, Robert  
 Duncan, Karen  
 Dunn, Robert  
 Dutton, William  
 Duvall, Lorraine M.

Eberhart, Charles R.  
 Eberly, Bill  
 Edwards, Gwen  
 Egerman, Paul L.  
 Eifler, Thomas A.  
 Embley, David W.  
 Engel, Gerald  
 Enzer, Selwyn  
 Estabrook, Leigh  
 Evans, Roger

Fairley, Richard  
 Faught, Williams S.  
 Feldstein, M. Alan  
 Felskin, Jerry  
 Fife, Dennis  
 Finegold, Joseph  
 Firth, Robert  
 Fischer, Kurt F.  
 Fischler, M. A.  
 Fisher, Grant

Fisher, Paul  
 Fong, Elizabeth  
 Force, Gordon  
 Forman, Joel Jon  
 Ford, William H.  
 Frazier, William H.  
 Fujii, Marilyn  
 Fujiwara, Atsumu

Gallant, William R.  
 Gallegos, Frederick  
 Garcia, Hector  
 Gates, Terry  
 Gentleman, W. Morven  
 Gerhart, Susan  
 Gilb, Tom  
 Gilbert, Barry K.  
 Glass, Robert L.  
 Goodlett, James C.  
 Goodman, Seymour  
 Gordon, M. E.  
 Gorg, Roger J.  
 Green, Edward  
 Greenberg, Don  
 Grey, Richard  
 Groppa, Richard  
 Gupta, Y.  
 Gustafson, G. G.

Hales, Blaine  
 Hammer, Michael  
 Harada, Minoru  
 Hardy, Viann B.  
 Harper, Bobby D.  
 Harris, Larry  
 Hart, Hal  
 Harvey, Bob  
 Hazan, Paul  
 Heafner, John F.  
 Heath, Fred  
 Heimbigner, Dennis  
 Heines, Jesse M.  
 Hendrix, Gary  
 Herlevich, F. Ann  
 Hewitt, Peter  
 Hilborn, Gene  
 Hilfinger, Paul  
 Hinerman, Frank  
 Hofkin, Bob  
 Holden, Jeffrey B.  
 Holland, John F.  
 Horn, Berthold K. P.  
 Horne, P.  
 Howell, Thomas A.  
 Hsia, Pei

- Hunt, Garry G.  
Hurvitz, Josh
- Ingrassia, Frank S.  
Irani, Keki B.  
Iwayama, Masatoshi
- Jacks, Ed  
Jacobs, Steven M.  
Jefferson, David  
Jensen, A. P.  
Jensen, Randall  
Johansen, Robert  
Johnson, T. A.  
Jordan, Jim  
Joslin, Edward O.
- Kanner, Mel  
Kapur, R. N.  
Karbach, Dennis  
Karplus, Walter  
Kartashev, Steven  
Kartashev, Svetlana  
Katzman, James  
Kaufman, Leon  
Keen, Peter  
Ketchel, James  
Key, G. S.  
Kimbleton, Steve  
King, John  
King, R.  
Kirningham, Brian  
Klassen, Daniel  
Klinzak, Stan  
Knowles, R.  
Konsynski, Benn R.  
Koyama, Kenji  
Kraemer, Kenneth L.  
Kraft, Philip  
Krzysiak, Emily A.  
Kunecke, Harold  
Kunii, Toshiyasu L.  
Kunin, Jay S.
- Ladd, I.  
Lasden, Martin  
Lee, Anthony P.  
Lee, John A. N.  
Levy, Stephen R.  
Lieberman, Mark A.  
Lindstrom, Dean  
Lipka, Steven  
Lipovski, G. J.  
Littlewood, B.  
Lobel, Jerome  
Love, Hubert H., Jr.  
Lowenthal, Eugene  
Luke, John  
Lytle, Ben
- Madeous, Richard  
Madison, William  
Maitlen, Richard  
Malek, M.  
Manley, John H.  
Mannoni, Michel  
Mariani, Michael D.  
Marill, Thomas  
Markus, M. Lynne  
Martens, Jon  
Maryanski, Fred J.  
Masatello, Robert  
Mason, Ida  
Matheny, Charles  
McClung, D. B.  
McCullough, Timothy  
McGill, Michael J.  
McKissick, Jack  
McLeod, Dennis  
McTap, John L.  
Meenaghan, Eamon  
Merrill, O.  
Merwin, Richard E.  
Migneault, Earle  
Miller, Gene I.  
Mirhakak, Mohammed  
Mitchell, Joan L.  
Moler, Cleve  
Moranda, Paul B.  
Mori, F.  
Morgan, William A.  
Morris, James  
Moursund, David  
Mulhall, Brendan D. L.  
Munnecke, Thomas  
Musa, John
- Naumann, J. David  
Navlakha, Jainendra  
Neilson, R. F.  
Nezu, Kohi  
Nickerson, Devon  
Nielsen, Francis H.  
Nilles, Jack  
Nottage, Richard  
Nudd, Graham R.  
Nycum, Susan H.
- O'Donnel, Joe  
O'Kelley, Harold E.  
Ogawa, Yutaka  
Ormancioglu, Levent  
Orsaza, Al  
Ouellette, Eugene  
Owens, Donald P.
- Palmer, John F.  
Panzl, Dave  
Parikh, Shyam
- Parks, Judith A.  
Parnas, David  
Peters, John K.  
Peters, Lawrence J.  
Peterson, Duwane  
Pflager, Richard  
Pitt, Paul  
Plagman, Bernard  
Pramanik, Sakti  
Premkumar, U. V.  
Presser, Leon  
Principato, Gene  
Pyster, Arthur B.
- Raber, David D.  
Rammamoorthy, C. V.  
Rannow, Robert L.  
Rassmussen, Terry  
Rault, J. C.  
Rice, Rex  
Riddle, Kay  
Rind, Kenneth W.  
Robb, Richard A.  
Robertson, Michael  
Robinson, Richard A.  
Robinson, W. B.  
Roman, Gruia-Catali  
Rosenblatt, Judah  
Rosenthal, Paul H.  
Rothnic, James  
Rotolo, Elio  
Roufa, Sheldon P.  
Roush, C. Steven  
Ruchinskias, J. E.
- Sagues, Paul  
Sandewall, Erik  
Santhanam, V.  
Sastry, K. V.  
Sawyer, Gary  
Sayani, Hasan H.  
Scacchi, Walt  
Scalf, Joseph  
Schiebe, Lowell H.  
Schmidt, Larry G.  
Schwarm, Stephen  
Sealy, David  
Sebrell, William  
Sejnowski, M. C.  
Selinger, Patricia Griffiths  
Shaiken, Harley  
Shaw, Ward  
Shiveley, M.  
Shooman, Martin  
Shortley, John  
Shosa, Dale  
Signor, Richard W.  
Simmons, Dick B.  
Smetanka, Terence D.

Smith, David Ned  
Smith, Dudley C.  
Smith, John  
Smoliar, Stephen W.  
Snyder, David  
Stearns, Robert H.  
Steinberg, David L.  
Steinfeld, C. W.  
Stevenson, D. K.  
Stewart, Robert G.  
Stonebraker, Mike  
Stover, Philip C.  
Stucki, Leon  
Su, Stanley  
Sulg, Madis  
Svenning, L. L.  
Swanson, E. Burton  
Swearingen, John  
  
Tai, Kuo-Chung  
Takahashi, Naohisa  
Takeshita, Tomomichi  
Tanniru, Mohan R.  
Tasker, Peter  
Tausworthe, Robert  
Teichroew, Daniel  
Tenenbaum, Jay M.

Terchritzis, D.  
Tharp, Alan L.  
Theis, Douglas J.  
Tomanek, Jerry  
Topkus, Raymond  
Troost, Marcus  
Turtle, Sherry  
Turn, Rein  
  
Ulloa, Miguel  
Ulrich, Walter E.  
Umeh, Fidelis  
Upchurch, E. T.  
  
Vallone, Antonio  
Vaughn, Carol  
Volland, Paul M.  
von Meister, William F.  
von Urff, Charles  
  
Wagner, Robert F.  
Wah, Benjamin W.  
Walker, Stephen T.  
Wallace, Robert J.  
Wallis, Robert H.  
Walstrom, John  
Ware, Willis

Warner, Ben, Jr.  
Wasserman, Anthony  
Wecker, Stewart  
Weingarten, Fred  
Weinstein, Bertram S.  
Weisbein, Marvin  
Wetmiller, John R.  
White, John W.  
Whitney, John  
Wiederhold, Gio  
Wilder, William L.  
Wing, Roger W.  
Wohl, Amy  
Wolf, H. C.  
Wolverton, Ray  
Work, Anne  
  
Yao, S. Bing  
Yeh, Raymond  
Young, Stephen  
Yourdon, Edward  
  
Zachman, John A.  
Zatyko, Dan  
Zawacki, Robert A.  
Zolnowski, Jean C.

# AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES, INC. (AFIPS)

## OFFICERS

### *President*

Albert S. Hoagland  
IBM Corporation  
San Jose, CA

### *Vice President*

Sylvia Charp  
The School District of Philadelphia  
Philadelphia, PA

### *Treasurer*

M. Stuart Lynn  
University of California  
Berkeley, CA

### *Secretary*

E. Ronald Carruth  
Minnesota School District  
St. Paul, MN

### *Acting Executive Director*

Jerry Koory  
AFIPS  
Arlington, VA

## BOARD OF DIRECTORS

### *Association for Educational Data Systems (AEDS)*

Judith B. Edwards  
Northwest Regional Educational Lab.  
Portland, OR

### *American Institute of Aeronautics & Astronautics (AIAA)*

H. Lewis Parker  
COMSAT Labs.  
Rockville, MD

### *American Society for Information Science (ASIS)*

Harold Borko  
University of California  
Los Angeles, CA

### *American Statistical Association (ASA)*

George Minich  
World Bank  
Washington, DC

### *Association for Computational Linguistics (ACL)*

Donald E. Walker  
SRI International  
Menlo Park, CA

### *Association for Computing Machinery (ACM)*

Daniel McCracken  
Ossining, NY  
Raymond E. Miller  
IBM T. J. Watson Research Center  
Yorktown Heights, NY

Aaron Finerman  
University of Michigan  
Ann Arbor, MI  
Steven S. Yau  
Northwestern University  
Evanston, IL

Tse-yun Feng  
Wright State University  
Dayton, OH

### *Instrument Society of America (ISA)*

Arthur C. Lumb  
Proctor & Gamble Company  
Cincinnati, OH

### *Data Processing Management Association (DPMA)*

George Eggert  
Chicago Dept. of Defense  
Chicago, IL

Robert Marrigan  
Mail Communications, Inc.  
Everett, MA

J. Ralph Leatherman  
Hughes Tool Company  
Houston, TX

### *IEEE-Computer Society*

Rolland B. Arndt  
Sperry Univac  
St. Paul, MN

### *Society for Computer Simulation (SCS)*

Per Holst  
The Foxboro Company  
Foxboro, MA

### *Society for Industrial and Applied Mathematics (SIAM)*

Donald L. Thomsen, Jr.  
SIAM Institute for Mathematics and  
Society  
New Canaan, CT

### *Society for Information Display (SID)*

Carlo P. Crocetti  
Rome Air Development Center  
Griffis Air Force Base, NY

### *AFIPS Immediate Past President*

Theodore J. Williams  
Purdue University  
West Lafayette, IN

## AFIPS EXECUTIVE COMMITTEE

Albert S. Hoagland  
IBM Corporation  
San Jose, CA

Sylvia Charp  
The School District of Philadelphia  
Philadelphia, PA

M. Stuart Lynn  
University of California  
Berkeley, CA

E. Ronald Carruth  
Minnesota School Districts  
St. Paul, MN

Aaron Finerman  
University of Michigan  
Ann Arbor, MI

Robert Marrigan  
Mail Communications, Inc.  
Everett, MA

Steven S. Yau  
Northwestern University  
Evanston, IL

Per Holst  
The Foxboro Company  
Foxboro, MA

## NATIONAL COMPUTER CONFERENCE BOARD MEMBERS

*Chairman and DPMA Representative*

Robert Marrigan  
Mail Communications, Inc.  
Everett, MA

*Vice Chairman and SCS Representative*

Carl Malstrom  
North Carolina State University  
Raleigh, NC

*Treasurer and AFIPS Representative*

M. Stuart Lynn  
University of California  
Berkeley, CA

*AFIPS Representatives*

Albert S. Hoagland  
IBM Corporation  
San Jose, CA

Sylvia Charp  
The School District of Philadelphia  
Philadelphia, PA

Harold Borko  
University of California  
Los Angeles, CA

*Secretary and IEEE-Computer Society Representative*

Dick B. Simmons  
Texas A & M University  
College Station, TX

*ACM Representative*

Seymour Wolfson  
Wayne State University  
Detroit, MI

*SCS President*

Stewart I. Schlesinger  
The Aerospace Corporation  
Los Angeles, CA

*DPMA President*

Robert A. Finke  
Cummings Engine Co.  
Columbus, IN

*ACM President*

Daniel McCracken  
Ossining, NY

*IEEE-Computer Society President*

Tse-yun Feng  
Wright State University  
Dayton, OH

*Chairman of the Industry Advisory Panel*

Frederick M. Hoar  
Fairchild Camera and Instrument Co.  
Mountain View, CA

*Chairman of the NCC Committee*

Jerry Koory  
On-Line Business Systems, Inc.  
San Francisco, CA

*Chairman of the Ancillary Activities Committee*

Jack Sherman  
Lockheed Missiles and Space Corp.  
Sunnyvale, CA

## NATIONAL COMPUTER CONFERENCE COMMITTEE OF THE NCC BOARD

*Chairman*

Jerry Koory  
On-Line Business Systems, Inc.  
San Francisco, CA

*Secretary*

Morton M. Astrahan  
IBM Research Laboratory  
San Jose, CA

*NCC '80 Chairman*

Herbert B. Safford  
GTE Data Services, Inc.  
Marina Del Ray, CA



*NCC '81 Chairman*

Al Hawkes  
Sargent & Lundy Engineers  
Chicago, IL

Russell K. Brown  
Moore Paper Companies  
Houston, TX

Harvey L. Garner  
University of Pennsylvania  
Philadelphia, PA

Floyd Harris  
Life of Georgia  
Atlanta, GA

Edward V. Resta  
Manhattan Beach, CA

Irwin J. Sitkin  
Aetna Life and Casualty  
Hartford, CT

Arnold P. Smith  
IBM Corporation  
White Plains, NY

Robert C. Spieker  
AT&T  
New Brunswick, NJ

Jeffrey D. Stein  
On-Line Business Systems, Inc.  
San Francisco, CA

## NATIONAL COMPUTER CONFERENCE BOARD INDUSTRY ADVISORY PANEL

*Chairman*

Frederick M. Hoar  
Fairchild Camera and Instrument Co.  
Mountain View, CA

Gordon Daggy  
Fairchild Camera and Instrument Co.  
Mountain View, CA

Al Ericson  
Dataproducts Corporation  
Woodland Hills, CA

S. A. "Sandy" Lanzarotta  
Xerox Corporation  
El Segundo, CA

William Lonergan  
Xerox Development Corporation  
Beverly Hills, CA

Richard Mau  
Sperry Rand Corporation  
New York, NY

Herbert Richman  
Data General Corporation  
Westboro, MA

Gordon Smith  
Memorex Corporation  
Santa Clara, CA

Dallas Talley  
Qantel Corporation  
Hayward, CA

## NATIONAL COMPUTER CONFERENCE BOARD ANCILLARY ACTIVITIES COMMITTEE

*Chairman*

Jack Sherman  
Lockheed Missiles and Space Corp.  
Sunnyvale, CA

Stanley Winkler  
IBM Corporation  
White Plains, NY

## AFIPS HEADQUARTERS AND CONFERENCE SUPPORT STAFF

*Acting Executive Director*  
Jerry Koory

*Secretary*  
Linda Kowalski

*Manager Administrative Support Services*  
Jane Smith

*Secretary*  
Patricia Mayo

*Accountant*  
Michael Powers

*Bookkeeper*  
Melinda Yost

*Financial Secretary*  
Joyce Paige

*Public Information Coordinator*  
Nancy Lefebvre

*Secretary*  
Betty S. Foley

*AFIPS Press Marketing Manager*  
Christopher Hoelzel

*AFIPS Press Fulfillment Administrator*  
Olive Shilland

*Secretary*  
Cynthia Carter

*AFIPS Press Editorial/Production Specialist*  
Ellen Marie Randall

*Director, AFIPS Washington Office*  
Alexander D. Roth

*Secretary*  
Kelly Andrus

*Administrative Assistant*  
Pearl Cook

*Research Associate*  
Ellen Law

*Assistant Research Associate*  
Keith Yankow

*Director of Conferences*  
Gerard Chiffriller

*Secretary*  
Nancy Betz

*Associate Director of Conferences*  
Carol Sturgeon

*Marketing Coordinator*  
Kate Frye

*Manager Exhibit Sales*  
Marie Stewart

*Secretary*  
Victoria Braskett

*Manager Conference Operations*  
Sam Lippman

## AUTHOR INDEX

- Albright, Thomas G., 121  
Amamiya, Makoto, 147  
Anderson, Mary Pastel, 441  
Appleton, Daniel S., 307
- Baird, George N., 811  
Barbacci, Mario, 209, 219, 229  
Barrow, H. G., 391  
Baumann, L. S., 549  
Baxter, Brent, 437  
Beamer, Gary D., 479  
Berlinger, Eli, 773  
Bolles, R. C., 391  
Borden, Chester S., 341  
Borrione, Dominique, 209, 219, 229  
Bowen, John B., 697  
Bracker, W. E., 41  
Braun, Christine L., 465  
Brown, David G., 441
- Capitant, Patrice J., 415  
Chang, Hsu, 191  
Charlu, Daniel P. S., 631  
Cochran, John E., 509  
Colton, Kent W., 841  
Comer, Douglas, 807  
Cook, Carolyn L., 555  
Coop, R. D., 549  
Copeland, George, 191  
Courtney-Saunders, Susan, 587  
Crane, Steven N., 73
- Davis, Richard B., 335  
de Valpine, Jean E., 785  
Deitrick, William R., 83  
Dietmeyer, Donald, 209, 219, 229  
Duvall, Lorraine, 677
- Eifler, Thomas A., 55  
El-Masri, Ramez, 319  
Embley, David W., 301  
Epstein, Robert, 237
- Faught, William S., 459  
Finegold, Joseph G., 329  
Fischler, M. A., 391  
Fisher, Paul, 191  
Fong, Elizabeth, 261  
Fujiwara, Atsumu, 113
- Gale, John, 87  
Gilbert, Barry K., 427  
Goodlett, James C., 503  
Gordon, M. E., 597  
Gorg, Roger J., 811  
Gupta, Y., 347  
Gustafson, G. G., 741
- Hammer, Michael, 541  
Harada, Minoru, 33  
Hawthorn, Paula, 237  
Heafner, John F., 855, 863  
Heimbigner, Dennis, 283  
Hektor, Göran, 569  
Herlevich, F. Ann, 329  
Hilborn, Gene, 157  
Hill, Fredrick, 209, 219, 229  
Hohn, William C., 129  
Holden, Jeffrey B., 493  
Horn, Berthold K. P., 371  
Horne, P., 643
- Iwayama, Masatoshi, 113
- Jacobs, Steven M., 687  
Johnson, T. A., 741
- Kapur, R. N., 623, 631, 643  
Kartashev, Steven I., 165  
Kartashev, Svetlana P., 165  
Kasper, Jerome V. V., 335  
Kaufman, Leon, 445  
Key, G. S., 741  
Kimbleton, Stephen R., 261  
King, Rob, 87  
Knowles, R., 347  
Konsynski, Benn R., 41  
Koyama, Kenji, 147  
Kraemer, Kenneth L., 841  
Krzysiak, Emily A., 671  
Kunii, Toshiyasu L., 33  
Kunin, Jay S., 541
- Ladd, Ivor, 533  
Lipovski, G. J., 623, 631, 643  
Littlewood, B., 707  
Lobel, Jerome, 831  
Love, Hubert H., Jr., 181  
Lowenthal, Eugene, 191
- Malek, M., 643  
Martens, Jon, 677  
Maryanski, Fred J., 293  
McCullough, T. L., 409  
McGill, Michael J., 683  
McLeod, Dennis, 283  
McTap, John L., 767  
Merrill, O., 347  
Merwin, Richard E., 139  
Migneault, Gerard E., 715  
Miller, Gene I., 795  
Mirhakak, Mohammed, 139  
Mitchell, Joan L., 423  
Moler, Cleve, 363  
Mori, Fumihiko, 1  
Mulhall, Brendan D. L., 687  
Munnecke, Thomas, 723
- Navlakha, Jainendra K., 871  
Nezu, Koji, 257  
Nickerson, Devon, 827  
Nielsen, Frances H., 855, 863  
Nudd, Graham R., 377  
Nycum, Susan H., 587
- O'Kelley, Harold E., 499  
Ogawa, Yutaka, 147  
Ormancioglu, Levent, 817  
Overgaard, Mark, 747
- Palmer, John F., 887  
Parks, Judith A., 811  
Pflager, Richard C., 791  
Piloty, Robert, 209, 219, 229  
Pramanik, Sakti, 837  
Premkumar, U. V., 623, 643
- Raber, David D., 13  
Reed, Martin A., 105  
Rind, Kenneth W., 795  
Robb, Richard A., 427  
Robinson, Richard A., 671  
Robinson, W. B., 597  
Roman, Gruiia-Catalin, 269  
Rosenthal, Paul Herbert, 613  
Roush, C. Steven, 293  
Ruchinskas, John E., 895
- Sandewall, Erik, 569  
Santhanam, Viswanathan, 877  
Sagues, Paul, 607

- 
- Sastry, K. V., 453  
Sato, Takashi, 1  
Schiebe, Lowell H., 135  
Schuster, Stewart, 191  
Sejnowski, Matthew C., 631  
Shiveley, M. Wayne, 863  
Shosa, Dale, 445  
Simmons, Dick B., 757  
Skelly, Patrick, 209, 219, 229  
Slater, Dan, 87  
Smetanka, Terence D., 105  
Smoliar, Stephen W., 67  
Snyder, R. David, 51, 73  
Sørensen, Henrik, 569  
Steinberg, David L., 883  
Steinfeld, Charles W., 895  
Stevenson, David K., 357  
Stock, Bruce E., 473  
Stover, Philip C., 51  
Ström, Anders, 569  
Strömberg, Claes, 569  
Strömfors, Ola, 569
- Su, Stanley, 191  
Svenning, Lynne L., 895
- Tai, Kuo-Chung, 275  
Takahashi, Naohisa, 147  
Tanniru, Mohan R., 23  
Tenenbaum, J. M., 391  
Tsichritzis, D. C., 533  
Tharp, Alan L., 275  
Theis, Douglas J., 93  
Tomanek, Gerald, 527  
Tsuji, Hiroshi, 1  
Turn, Rein, 581
- Ulloa, Miguel A., 473  
Ulrich, Walter E., 485, 489  
Upchurch, Edwin T., 631  
Urmi, Jaak, 569
- Vallone, Antonio, 801
- Wagner, Robert F., 441  
Wah, Benjamin W., 243  
Walker, Stephen T., 655  
Wallace, Robert J., 121  
Wallis, Robert H., 415  
Wasserman, Anthony I., 731  
Wetmiller, John R., 7  
White, John W., 515  
Wiederhold, Gio, 319  
Wilder, William L., 823  
Wolf, H. C., 391
- Yao, S. Bing, 243  
Young, S., 347
- Zolnowski, Jean, 757