multi-access computing: modern research and requirements

Edited by PAUL H. ROSENTHAL & RUSSELL K. MISH

system development corporation

HAYDEN

Rosenthal • Mish

MULTI-ACCESS COMPUTING

HAYDEN 5964-7

$13.95

MULTI-ACCESS COMPUTING:
Modern Research and Requirements

Edited by Paul H. Rosenthal & Russell K. Mish, both of System Development Corporation

No longer the luxury of the more elite computer facilities, multi-access computing has become widely prevalent, and its applications and advantages are growing.

This comprehensive volume embraces the latest application requirements as well as some of the most important research now shaping multi-access capabilities for the future. It is an indispensable tool for all users and potential users of interactive systems.

Developed from the proceedings of a "by invitation only" symposium sponsored by the Office of Naval Research and System Development Corporation, the book gives readers a first-hand view of the work of leading ADP users and developers. More than a survey, it can also serve as a practical guide for formulating individual application requirements and forecasting the future availability of data processing capabilities.

Part I covers scientific analysis computation, process control systems, administrative data processing systems analysis, and other applications.

Part II outlines the work of three outstanding applications laboratories: SDC's ARPA network resources, automated programming, associative processors, and graphic reporting. The RAND Corporation's interactive graphic console, microprogramming facility, and other activities. The Stanford Research Institute's Artificial Intelligence Robot and Large File Management Program activities.

(continued on back flap)

(continued from front flap)

Part III puts the reader right in the midst of crucial projects under way at Bell Labs, MIT, The Mitre Corporation, California Institute of Technology, the USAF, and other key institutions.

Part IV concludes this excellent volume with a discussion of policy considerations for the future.

Paul H. Rosenthal was selected to plan and organize the symposium based on his broad data processing, administrative, and education experience. He is a system consultant with System Development Corporation. Consultant and co-editor Russell K. Mish is a proposal specialist at System Development Corporation.

MULTI-ACCESS COMPUTING

MULTI-ACCESS COMPUTING

Modern Research and Requirements

Edited by

PAUL H. ROSENTHAL

and

RUSSELL K. MISH

System Development Corporation
Santa Monica, California

HAYDEN BOOK COMPANY, INC.
Rochelle Park, New Jersey

Library of Congress Cataloging in Publication Data

Main entry under title:

Multi-access computing: modern research and requirements.

Selected papers of a conference jointly sponsored by System Development Corporation and the Office of Naval Research.
1. Time-sharing computer systems—Congresses. 2. Real-time data processing—Congresses. I. Rosenthal, Paul H., ed. II. Mish, Russell K., ed. III. System Development Corporation. IV. United States. Office of Naval Research.
QA76.53.M84 001.6'44'04 74-4067
ISBN 0-8104-5964-7

Hayden Book Company, Inc.
50 Essex Street, Rochelle Park, New Jersey 07662

Printed in the United States of America

The research reported by these Proceedings was sponsored by the Office of Naval Research under Contract No. N00014-71-C-0293, Contract Authority Identification No. NR-049-318/2-8-71 (437). Reproduction in whole or in part is permitted for any purpose of the United States government.

1 2 3 4 5 6 7 8 9 PRINTING

74 75 76 77 78 79 80 81 82 YEAR

Foreword

On behalf of the Office of Naval Research (ONR), I want to thank System Development Corporation for the time and effort spent in organizing and planning the conference, for their willingness to listen to our ideas on how it should be conducted, and for their ability to reach a reasonable compromise. As for the audience, I would like to thank them for their stamina in sitting through three days of intensive preparations and discussions that probably should have been extended to four days to allow sufficient time for the papers and discussions.

Several times during the conference I have been asked why ONR sponsors meetings of this type. Many see ONR's mission simply as support of research projects. It really is not to anyone's advantage to just sponsor research and then let it lie buried in a technical journal, yet all too frequently that is what happens. Therefore, although most of our funds are used to sponsor research, we also try to act as a catalyst for the implementation of research into government and/or industrial operations. Symposiums such as this help disseminate the research and allow feedback from the potential user to the scientist.

ONR attempts to sponsor symposiums on topics that have not had extensive exposure. When SDC presented the concept of a symposium to discuss current and future research activities in multi-access computing, we thought it was a good topic. Dr. Thompson comments in his overview that there is a great lack of congruity in the papers presented. That is the very reason why a meeting such as this is so valuable. If everyone understood the role and importance of each others work, and the research activity was already integrated, there would be no need for these types of symposia.

---

*This foreword is derived from the closing remarks to the symposium delivered by Mr. Goldstein.

There is also a void in the government research establishments of today that we are attempting to rectify with meetings of this type. There is no DOD agency that I know of whose job it is to take the results of research, initiate pilot development, and then, if successful, to sponsor introduction into DOD operations. The solution to the implementation problem today is to have the help of everyone concerned. For example, the designers advertise their work at these meetings, and the potential users, at the same time, look to them for solutions to their identified problems. Often a company will see the utility of the research and decide to contribute to its further development. In this fashion, numerous ONR projects have resulted in government equipments and commercial products. Sometimes when research is applied by this indirect method, the Navy does not get credit for the original sponsorship, nevertheless our society does benefit and research money is considered well spent.

Gordon D. Goldstein
Information System Program
Office of Naval Research

Preface

Distributed data processing systems – bringing the computer system interface into the field or remote facility – represent the most sophisticated form of information systems. Once associated with very large, very expensive real-time systems, the distributed data system is now practical for modest investment and operating costs. This book presents the proceedings of a conference dedicated to advancing the state-of-the-art in multi-access computing systems – one of the two basic approaches being used to implement distributed information systems (the other basic approach is decentralized computing using minicomputer networks). The conference was planned to bring together in a reasonably sized working group (total attendance was planned at 100 invitees) leading computer scientists, key government EDP users, and key university researchers to maximize communication. Response to the conference was extremely favorable, and the interest in making widely available at least the formal portions of the conference has led to the publication of this book.

This collection seeks to present the requirements to be imposed on multi-access computing: timesharing, real-time, and remote batch, and includes current research. The purpose is to improve the communication between the researcher and the user. The focus is primarily on military type applications but most of the material is equally useful for industrial purposes. Several papers specifically address consumer and commercial applications.

By design, divergent views and ideas are included. Two commentary articles, for example, discuss divergent views on several other papers. Requirement papers range from detailed specifications to problem presentations. Research reports range from detailed results of specific tasks to surveys of available and planned equipment. We hope the reader will gain

from this range in the same way that the variety stimulated interchange between the participants.

To best present the views of the various participants, we have attempted to transcribe each paper as it was given, and edit only to the extent necessary to assure continuity of thought.

In a heterogeneous collection of this type it is difficult to adequately express our appreciation to the many people who contributed. We would like to express our sincere appreciation to the contributing authors who have been kind enough to permit the inclusion of their speeches. And, of course, our greatest debt is to Mr. Marvin Denicoff and Mr. Gordon Goldstein of the Office of Naval Research whose advice, encouragement, and support made the conference and these proceedings possible.

Acknowledgments

The editors wish to acknowledge the work of the following individuals whose assistance contributed materially to this book:

Mr. Gordon D. Goldstein of the Office of Naval Research who co-hosted the symposium at which the accompanying papers were presented.

Mr. James Copes of System Development Corporation who performed the myriad administrative tasks necessary to hold a symposium of this type.

Mr. Ronald L. Citrenbaum and Mr. Lawrence H. Guthrie of System Development Corporation who edited numerous papers and assisted in assembling the initial draft of this document.

Contents

MULTI-ACCESS COMPUTING

1. Multi-Access Computing in the 70s

Paul H. Rosenthal

System Development Corporation

Santa Monica, California

As a confirmed optimist, I forecast that the 70s will be the decade of "successful systems." Successful systems may be defined by the following set of criteria presented in a recent AMA publication.*

Relevance - meeting the actual needs of the organization.

Timeliness - meeting the natural cycle times of the total business system.

Economy - meeting basic tangible cost/value criteria and not performing a luxury function.

Flexibility - meeting growth and on-demand requirements.

Accuracy - maintaining correct data bases and performing auditable processing.

A large proportion of the military and commercial systems involving computer-related data processing will require reimplementation during the 70s to meet these "success" criteria. Lower cost computer hardware will make this technically possible, often by bringing the system into the office, where the user interfaces with the environment. This requires distributed data processing, an area that is expected to reach 50 percent of total EDP activities by the late 70s. Distributed data processing is being implemented

---

*Burton J. Cohen, Cost-Effective Information Systems, American Management Association, New York, 1971, pp. 13-16.

through two technologies: (a) multi-access computing (remote use of a central processing facility) – the topic of this volume, and (b) decentralized computing – through extensive utilization of minicomputing methods.

Multi-access computing generally involves the use of remote batch methods (RJE), interactive processing (time sharing), and real-time computing (process control). These methods are often the only way of meeting the successful system criteria. For example:

> Remote Batch Methods - Normally used for large processing jobs. When the data originate at multiple locations, but a centralized data base is required, only this method is: relevant – meets user needs; timely – uses an up-to-date data base; and accurate – allows controls at the data origination point.
>
> Interactive Processing - Normally used for small to medium processing jobs. When utilization is not steady at each individual site, or some requirement of the processing exceeds the capabilities of the minisystem, the method is: economic – spreads computing over all sites; and flexible – more users and applications can be handled on a more flexible schedule.
>
> Real-Time Computing - Normally used for process control of systems involving multiple functions and sites. The man/machine/process systems currently so popular can normally only be performed using multi-access methods.

Multi-access computing is, therefore, a generic term for the communication based systems required for the operational, scientific, and administrative applications of the 70s. It is a large segment of the total EDP developments now being considered and planned. Therefore, what are the requirements imposed on the design, software, and hardware for such systems? And, what research is being performed that may meet these requirements? This volume, hopefully, answers these questions.

PART I. COMPUTATION REQUIREMENTS

The papers in this section are oriented toward the requirements for operational and support applications – trying to provide systems and technology to meet user demands first and scientists' interests second. That the first section comprises projected requirements indicates the recognition of the importance of relevance – not only to the needs of today, but to the anticipated needs of the decade ahead. The problem is not to solve yesterday's problems with today's tools, but to anticipate needs sufficiently to produce data processing systems for tomorrow's then current problems with then available tools.

It is often pointed out that to understand the problem is to be halfway to the solution. Much of the problem is in identifying the user, his need, and his deadline. For the last two decades when these problems finally were defined, there was no practical method of solving them. The multi-access hardware and software now becoming available have changed this situation. Now, a great deal of what was impractical in the 60s will be practical in the 70s. It behooves us, therefore, to note carefully the users' views reflected in this section. Perhaps the 70s can become the era in which EDP actually meets user needs – i. e., the era of successful distributed computing systems.

The seven papers in this section reflect a cross-section of application requirements. Dr. Bergman presents weapon systems; Mr. Lamendola and Dr. Boehm present command and control systems; Dr. Pennington presents scientific processing; Mr. Swerling presents process control; Mr. Knepell outlines the major problem facing administrative users, and Mr. Kiviat closes with a discussion of system simulation requirements. The viewpoints cover most of the requirements imposed on multi-access computing and, except for a discussion of data base problems, all major problem areas are well covered.

2. Large Scale Computers vs. Real-Time Systems

C. E. Bergman

Naval Electronics Center
San Diego, California

Over the past twenty years, real-time systems have evolved largely from total use of analog components, with their inherent problems of stability and low precision, to the current, nearly universal use of digital techniques. Most of these digital systems utilize a centralized, large-scale, digital computer as the fundamental building block. The theoretical advantages of the digital computer are self-evident. Less obvious, however, is that costs associated with obtaining these advantages are rapidly becoming so great that the rationale for continued use of the large-scale computer in many real-time applications is becoming questionable.

Cost, in this context, means much more than the fixed cost associated with the procurement of the hardware and software packages of a given system. It refers to the price of generating functional programs from assemblers and compilers, a task often requiring a cadre of programmers and ADP software specialists, and the more subtle, but real, cost incurred because of the enormous complexity of the resultant system. This latter figure contains the cost which results from the loss of control over the total design by the system engineer (who usually doesn't comprehend the vagaries of software) and the subsequent reliance on programmers (who generally don't understand the technical details of the system) in order to make the system work. The result is an uneconomical and poorly engineered system designed to do a relatively straight-forward job.

A possible solution to this problem may exist from the analog days when system functions were implemented in hardware. Until recently, system implementation using digital components in this fashion was not feasible because of the high cost of digital logic and memory. Today, however, the cost of LSI batch processing is such that a flip-flop or a bit of memory can be obtained for approximately 1¢ and thus it is practical to consider utilizing hard-wired functional processors in lieu of a general-purpose machine with complex software algorithms.

Also important here is that current technology permits redefining the meaning of the term "digital computer." The general-purpose machine usually is considered a device with a single arithmetic unit which has time-shared interactions, based upon a complicated "interrupt" procedure, with I/O, control, and memory. This has led to the usual concept of "the computer," a concept no longer inviolate.

A LOOK AT THE ISSUES

The major issue associated with reorientation in the system concepts described above is flexibility. Clearly, there are numerous instances in which the flexibility afforded by software to effect major changes in the system operation justify the expenses. For many situations, however, this apparent flexibility is illusory. This is particularly true when the large-scale processor approaches time and/or memory limits in its operations. Under these circumstances, program changes are not easily accomplished for fear of violating these limits or causing a chain reaction as a result of program module interactions. In many cases, flexibility is not even required; e. g., control of external hardware, coordinate conversions, preprocessing of sensor signals, etc. For such situations, it seems sensible to consider alternative approaches to their implementation.

With the foregoing in mind, a number of possibilities for the design of real-time systems can be suggested. The conventional general-purpose machine, although commonly employed, is perhaps the least optimum solution, for the reasons stated. Multiprogramming and multiprocessing techniques being touted for use with such machines often tend to make the problem more complex and expensive. A variation on the general-purpose computer theme is the use of a set of distributed minicomputers. This approach offers many advantages including that the system is easier to design, test, maintain, and reconfigure. It also makes the system engineer the key individual in the system development.

If the traditional bond to the large-scale computer is broken by employing minicomputers as the controlling elements of "subsystems" of the real-time system, then there clearly is no fundamental reason for stopping at that point. In fact, current technology for the fabrication and design of digital logic from bipolar and MOS devices makes it exceedingly attractive to consider system implementation in terms of Direct Functional Mechanization (DFM). DFM defines a philosophy of system design, development, installation, checkout, and maintenance which is geared to today's technology and whose measure of performance can be stated in terms of the costs associated with each of these five factors. It is a philosophy based upon a sound understanding of system performance requirements and subsequent mechanization of the system by the use of functional modules constructed from components fabricated by the use of modern technological processes. With DFM as an additional

option available to the system designer, we have effectively removed the constraint of having only one tool, "the computer," to use in our system developments.

With the possibility of using DFM as a design tool, the process of system definition can now be stated in the following set of steps:

1. Examine desired system performance.
2. Examine necessary system functions.
3. Examine available tools (G. P., minicomputers, DFM, etc.)
4. Make tradeoffs.
   (a) cost/performance
   (b) hardware/software/firmware
   (c) digital/semidigital/linear
   (d) centralized/decentralized
   (e) custom design/off-the-shelf
   (f) dedicated/shared
5. Define system configuration.

In examples of the application of the DFM concept to several real-time processing systems being developed at NELC, the results to date conclusively suggest the potential value of this approach to design. If many of the other key issues such as off-the-shelf availability, keeping pace with technology, logistics, efficiency, size, weight, power, reliability, maintainability, availability, cost, manpower, training, and programming requirements are considered, it is obvious that many tradeoffs exist that need to be considered in choosing between the large-scale computer, minicomputers, or DFM for system implementation. The studies at NELC suggest that improvements of at least an order-of-magnitude might exist for most of these factors through the choice of DFM rather than the general-purpose machine. Even if the improvement is derated to 2:1, the DFM approach clearly merits consideration.

CONCLUSION

The DFM approach to systems design presents a number of weighty questions concerning the validity of continuing the conventional approach to the mechanization of real-time systems. By

utilizing DFM, the systems engineer can play a more extensive role in the total development, software complexity is drastically reduced and often largely eliminated, multiple simultaneous operations are possible as a result of architecture which avoids the restrictive single arithmetic unit of the general-purpose computer, and timing constraints and complicating system interactions are greatly alleviated.

3. Command/Control Requirements for Future Navy Systems

Michael A. Lamendola

Naval Electronics Laboratory Center

San Diego, California

There is no universally accepted definition of command and control. Generally, command and control is thought of as the processing of information culminating in an operational decision and as being computer oriented. There is processing and the information, after it is processed, is presented to a human being who, based upon the information presented, makes a decision to take an action or to take no action at all. Thus, the primary command and control requirement is reliability–reliability of the data that are gathered or the reliability of the processing and a high confidence value in the result. Unfortunately, today, command and control systems in many quarters are not viewed as being particularly reliable. In fact, the reliability of command and control systems seems to be going down, a solution that could have some grave consequences. This paper will examine how the predicament evolved and then will examine potential solutions to current and future command and control system problems.

HARDWARE TRENDS

The trend in the past has been to rely on advances in hardware technology to solve problems. This presentation will look at how computers have evolved from a perspective slightly different from that normally used. The first-generation computer structure had a single set of registers which were used to perform both the arithmetic operations and the input/output operations. Either one or the other was done, but not both simultaneously. Systems designed for first-generation computers had such things as an input translation phase to get the program into the machine. After calculations were performed, it was necessary to go through an output translation phase to get the program out onto a printer. This was done to maximize use of the machine.

Second-generation computer systems made life easier. The main difference between first- and second-generation machines was a second set of registers that were used for input/output operations. Input/output and arithmetic operations now could be performed simultaneously. It took about two years for system software to catch up and actually start using this feature and then only in a limited fashion. Even today it is not used nearly as much as one would expect. It is still customary to write something out on disk or tape and check to see if it went out all right before continuing with processing.

With the advent of the third-generation machine, there were still arithmetic registers, and separate input/output registers, but now there was also a new set of registers called base-address registers. These permitted keeping track of more than one program that was residing within the main, or primary, memory simultaneously. In other words, multiprogramming was now possible. Because multiprogramming could be done efficiently without having to perform a lot of address calculations, machines now could be time-shared effectively. Although time-sharing was possible on second-generation machines also, it wasn't nice, it wasn't clean, and it wasn't efficient.

The reader will notice that, in terms of hardware advances, the machine organization and the system implications of that machine organization have been emphasized rather than the development from vacuum tubes to transistors to integrated circuits. The organization was the key and held the system impact. It seems fair to say that during development, design objectives were primarily to keep the machines general purpose, to make them usable across as broad a range of tasks as possible, and, with a major thrust in design, to make more efficient use of the machine itself. Computers are very expensive and the aim has been to keep that processor busy as much of the time as possible. In the first-generation system organization, it was obviously very inefficient to be able to do only one thing at a time, either calculations or input/output. In second-generation machines, the inefficiencies were not quite so obvious. Hardware monitoring devices had to be attached to determine what was going on internally. They showed a processor busy by itself about 40 percent of the time and not used about 60 percent of the time. The input/output units were busy by themselves about 50 percent of the time and the two of them performed simultaneously about 10 percent of the time. Someone soon concluded that it was much more efficient to have two or more programs in memory so that if one was performing input/output the other could make use of the processor. The thrust was machine efficiency.

Is this good or bad, or is it some of each? These design objectives evolved in the commercial world. The Navy has been the recipient of this thrust and these design objectives. From the perspective of command and control, the general-purpose computer seems by definition almost mismatched to any given task. It would

be extremely rare for a single task to precisely match the resources of a general-purpose computer. Tasks will be either too large or too small. If the computer resources are too large for the task, the result, in addition to the question of cost effectiveness, is that functions tend to be added to the system that ought not to be added. This is a management problem, not a technical problem, but it does happen. When computer resources are too small for the task to be performed, the situation is more serious. Something has to give and it will not be the computer. It was no accident that for years programs fit conveniently into 32,000 words of memory. The task in one way or another must be degraded. If the task is concerned with response times, then the response times are going to slow down. If process capabilities are involved, then less will be processed than is desired.

SOFTWARE TRENDS

In this computer-task relation, the computer remains very rigid and inflexible. What is not fixed and what is used to fit the task to the machine is the software. The philosophy said the task was to be fitted to the machine. It now seems more appropriate for the machine to fit the task. This current task-computer relation places a tremendous reliance on software. In view of this reliance, one would expect that if he reviewed the development of software tools – those advances in the software state-of-the-art designed to aid the programmer in the design and implementation of the command and control system – he would see a parallel in the software area correspondingly roughly to the advances in hardware technology. Unfortunately, this isn't true. In fact there is a great deal of justification for the very harsh statement that there have been essentially no advances in software development in the past dozen years.

Most advances in software have occurred in operating system areas. In addition to the assembly language and machine language, programmers in 1960 programmed in high-order languages such as FORTRAN II and COBOL. They divided their programs into subprograms or modules called subroutines. These were fed into the operating system and processed. The loader in the 1960 operating system performed a library search and linkage and it had the intelligence to make a comparison so that if the programmer wanted to substitute his own particular SIN routine, for example, the operating system would recognize this fact and not go out and retrieve the system SIN routine from the library. The programs were structured in relocatable form to give some flexibility to the system and there were tools such as trace routines, trap routines, and dump routines which could be called upon either automatically or deliberately by the programmer under a variety of conditions depending upon the system he was working under. When a system was being constructed, these were the fundamental tools a dozen years ago.

Today, programmers still use high-level languages. FORTRAN and COBOL are still very popular, although in the command and control world JOVIAL and CMS-2 are used. Using subroutine structures hasn't changed. And there are still operating systems that operate in a somewhat job shop environment, setting up the programs, searching things out, and linking them. The programs still tend to relocatability and there are debugging aids such as trace, trap, and dump.

Now, this is obviously oversimplified. There are very worthwhile specializing routines that exist here and there. Automated flowchart packages, for example, facilitate documentation, which is always a horrendous job. A number of specialized languages for various disciplines have been developed, and there are specialized applications programs such as computer-aided design for circuit and logic design. However, such simplifications are not that far off and there has not been much progress in the software area.

What does this mean? The unit of measure in the development of a command and control system is the labor hour, the programmer's time. Little has been done to make that labor hour effective. MIT has done a great deal of work in man-machine interaction, and it has broken new ground in this area. So have a number of others. In systems development at NELC, online interactive programming techniques are used wherever possible. However, the basic structure of the software that is produced has not changed. Consequently, the problems with it have not changed.

IMPACT OF TRENDS

This then is the computational environment. In command and control, system functions are increasing in complexity. The number of system functions of a given system has been increasing over the years. The requirements to integrate discrete systems have been increasing. More and more, these systems are required to talk to one another. We require the use of general-purpose computers that are a mismatch to the task, and we employ essentially the same software tools that were developed in 1960. If the way the system is developed is held constant, and if over a period of time system complexity increases, the result is a decrease in the reliability of an individual system. Since the labor hour is still one hour long and since many more functions now have to be incorporated into a given system, the time to develop this system has been increasing over the past years and consequently the cost increases. Therefore, it is not too surprising to meet more and more individuals in positions to determine whether or not to initiate systems who are very reluctant to embark on yet another computer-oriented command and control system.

SYSTEM INTERFACING

One problem area that requires help is intersystem communications, the capability to use data produced by someone else. This is a current problem, but it is also a future systems problem. It is a problem that develops as an after-the-fact situation. System A has been developed and is doing whatever it is supposed to do. When System B is developed, if the designers are aware that they are going to have to use, as inputs, outputs from System A, then they will design their system to accept this output. This may be difficult but it is frequently done. However, what really creates a problem is when System B has already been developed and has no knowledge of System A's existence. Subsequently, a requirement is levied upon one or both of these systems either to share data or to have one of the systems use as input the output from the other system. Although this is possible, one discrete system usually cannot accept and assimilate another system's data because the computer programs for the individual ADP systems manipulate their individual data elements in different ways. They use different formats or they are on different types of computers. Consequently, overcoming the problems may require the construction of extensive individually tailored interfaces for each of the systems.

As more and more systems have communication equipment associated with them, so that they tie into a communications network and are thus interconnected by data lines, this requirement to share data or exchange data is growing by leaps and bounds. Once again, the programmer must get the pieces to fit and at present he doesn't have very good tools to accomplish this. He needs a mechanism for describing the data very precisely. Files of data contain not only data but also control information, such as record lengths and pointers. A control item has significance according to its location in a file or record. This significance is defined by the conventions of the operating system or the users' program which controls the data. The receiving programmer must be able to distinguish control information from data items. He must be able to interpret the logical relationships of the data from the information that he receives. All of this information must be presented to him so that he can make the appropriate translation into his system in order to be able to use it.

Normally this information is not contained within the data file itself. It is contained within the documentation of the program of the generating programmer. It may be implicit in his program structure, or it may be described in detail and in depth in very formal documentation. In any event, it is usually inconvenient to obtain this information and once obtained it is difficult to use and requires a lot of work. A tool is needed which does not require a lot of work, one which allows the programmer to describe his data fully and explicitly and provides for passing the description easily

to anyone else who wishes to use the data. The description ought to be formal and it ought to be standardized.

DATA BASE TRANSFORMATION

Given that a formal and standardized system interfacing is available, one may then consider another tool. This would be a program which translates data from one form and format to another. It might be called a generalized data base transformer. This tool would process the data base of one computer system and produce the data base suitable in structure and format for another system. It would be driven by descriptions of both the source data base and the target data base. It would reform at data items; it would replace control information with the pointers, indices, record counts, and so forth, that would be required to access that data in the new environment. Now it isn't certain that this is, in fact, possible. There has been a great deal of work done on the data description language, particularly by the Data Base Task Group of the CODASYL Committee. However, the problem area that is being addressed is not one that is expected to disappear. It certainly would seem a very worthwhile area of investigation, since it is of deep programming concern and can have a negative impact on the system structure. This function could be relegated to the operating system, as a service similar to many other services that the operating system performs. It would simply be initiated as needed by the programmer. That, of course, would be ideal.

CONVERSION

Another long-term problem area is that of changing machines – the transferability or portability of software programs from one computer system to another. This long has been the subject of investigation and thought and it continues to be a very worthwhile area of investigation. One argument that has been proposed for the existence of high-order languages, in addition to shortening the time required for a programmer to produce a program, is that there is a high degree of machine independence within a high-order language. Consequently, one can move from machine to machine with relative ease. In practice this doesn't seem to be the case. Languages, or rather implementations of languages by different manufacturers, tend to be nonstandard by whatever standards have been defined. Manufacturers tend to embellish languages to take advantage of certain features of their own machines. Consequently, one encounters limits on program size; limits on the number of programmer-assigned names; limits on total number of source statements, total number of characters and statement identifications, and so forth. These will vary from machine to machine. Programs tend to be dependent on the particular operating

systems from which they were developed for library functions, error recovery procedures, overlay structures, and file management, and there are no standards for these program-to-operating system interfaces.

One encounters explicit types of dependencies. However, there are implicit tendencies that occur also and these are just as horrendous to overcome as explicit dependencies. In a recent conversion effort for a large program more than 95 percent of the program was written in a high-order language. Less than 5 percent was written in machine language, and that portion was very carefully isolated, very well documented, and even identified in terms of the high-order-language statement that would be used if the removal of the machine-language statements were desired. They were in machine language for efficiency at very critical points where the designer felt that they could not tolerate the slightest inefficiencies that might be produced by high-order-language compilation. On the surface this seemed like a beautifully done job, ideal for transferring from one machine to another. However, the price associated with moving that program from one machine to another ran into many hundreds of thousands of dollars. The cost was about $600,000 for both machine time and labor on a project whose total cost was about $4,000,000, from requirements analysis through implementation. The fault lay in the implicit dependencies contained in the program. The programmer knew the machine for which the program was originally constructed. He knew its organization, its structure, how it behaved, how many bits made up each character, and how many characters were in a given word. He knew exactly where he would be if he skipped five words or fifteen words. This sort of logic was threaded all through this very large problem. It was not done intentionally and it certainly was not done to make it more difficult to move from one machine to another. It just happened. The result required a major effort to change from one machine to another.

A popular approach for solving program transferability has been to address it from the point of view of compiler construction. Compilers are used in such a way that the same source program can produce code for a variety of different target machines. Another approach has been to attempt to structure very rigorous standards for a given language and attempt to enforce these standards. However, neither approach solves the problem of the implicit dependencies.

A recent report of work being done in Japan mentioned an attempt to develop a machine-independent software system. The intent was to develop an intermediate high-level language and a compiler which would translate a variety of source languages into this intermediate language. This concept was first proposed in the literature around 1956-1958 in the United States and it was called UNCOL, or Universal Computer Oriented Language. This country

did not proceed very far with UNCOL, but apparently in Japan at least one computer manufacturer actually completed a compiler that takes this intermediate language, which was standardized across Japanese manufacturers, and translates it into machine language for his machine. This compiler implementation ran very slowly or comparatively slowly when compared to other compiler techniques. Because of the value Japanese manufacturers were placing on fast compiler time, the machine independent approach seemed to be given very little emphasis. This seems unfortunate, for within the command and control environment, programs can have exceedingly long lives, lives that can bridge computers from one machine to another.

REUSABLE SOFTWARE

The problem of transferability of software programs across systems leads to the question of reusable software. Reusable here means the ability to take a deck of cards representing an existing program in a given source language and to insert the deck into another program or into another deck of cards and use it directly without concern about the language of the program or the original target machine. Ideally, one would like to be able to make use of any software or any functions previously programmed. However, as indicated earlier, this is extremely difficult to do. Software is presently very sensitive to environment; that is, the machine characteristics, the operating system, the data, the computer language, or the compiler technique. If any one of these is changed, the software becomes inoperable or severely degraded and the programmer must get in and make changes to make it work. Ideally, it's preferable not to worry about these things. In designing and implementing a command and control system, it would be ideal to identify all of the functions encompassed in that system, ending up with an elaborate library of proven software functions, plus the tools that draw from this library and incorporate the functions into the development. This would allow the designer to avoid reinventing the wheel for each function. He could concentrate on developing only those functions specific or unique to this application.

To make this idea workable, the fundamental structure of software itself must be questioned. It is necessary to question how things get put together, how they get linked, how they get identified, what constitutes a module, and what determines the interfaces between the modules. Maybe this simply describes a super-library capability; but, perhaps, it could more appropriately be described as a new computer language, a much more powerful language than anything known today. It does seem appropriate, in terms of the objective to be achieved, to question why things are done the way they are, to attempt to determine if there might not be better ways of doing them. Certainly, some very fundamental

breakthroughs will be required to achieve this end. It also seems that a sort of limit has been approached if these breakthroughs are not found. Consider where software or the whole computer science field would be if a given function was programmed, developed, or devised only once, and all efforts were expended simply by adding new functions.

TESTING

Part of the concept of reusable software is that each function has been proved in operation and therefore is one part of the system that need not be tested further. The goal should be contracts which dictate that software will be delivered error-free even for those who are issuing the contract. Most of the current literature indicates this is not possible; the typical large software program has so many paths going through it that at best one can achieve only a certain confidence level and, having achieved that, must be satisfied with that result and must not insist on error-free performance. That undoubtedly is true if one limits himself to doing things as they are presently being done. If one removes that restriction, then the question might validly be posed, Why can't one have error-free software? Certainly, to the ONR community, there should be no constraints to say that the way things are done must continue without change.

SUMMARY

To summarize, the primary requirement for command and control systems is reliability. In the past, the tendency has been to rely on advances in general-purpose computer technology to increase reliability. However, the complexity of the systems has increased at so fast a rate that system reliability has decreased rather than increased. Advances in software technology have been minimal in command and control over the past dozen years and the needs identified call for a fundamental reexamination of the basic design and implementation processes.

4. Command/Control Requirements for Future Air Force Systems

Barry W. Boehm

The RAND Corporation

This paper begins by summarizing some of the currently disclosable results of a recent Air Force study on what should be done in information processing to meet major Air Force command and control requirements in the 1980s. It concludes with a few items that the author hopes people will consider to make computing more of a science.

AIR FORCE INFORMATION PROCESSING

Air Force information processing is a very big and complex business. Current Air Force software projects cost almost a billion dollars a year. Some Air Force computing operations are primarily online management information systems, such as the huge Advanced Logistics System being developed now. Interactive graphics goes on a good deal within the Air Force intelligence operations, as in the current TIPI program. The laboratories – weapons, flight dynamics, materials, etc. – run huge calculations, performed on large computers. Air Force space operations have very high bandwidth sensors pumping data into computers.

Individually, these represent major challenges to any kind of computer design, development and maintenance job that one wants to do. Air Force command and control has even more difficult problems because it tries to do all four of these things together.

UNIQUE FACETS OF COMMAND AND CONTROL INFORMATION PROCESSING

Air Force command and control to some extent is an online management information system; to some extent it involves interactive graphic manipulations; to some extent it involves large calculations for doing route planning or operational plan optimization

and the like; and to some extent it involves very high bandwidth sensor data processing for warning information. Doing all of these things together creates a huge challenge for the integrated design of a computer-based system.

However, several other systems outside of military command and control do this too. The air traffic control system and the NASA Apollo system also try to integrate all of these things and are very complex. However, command and control in the Air Force, Navy, or Army is even more difficult than this because of three additional considerations.

One consideration is the unpredictable environment. For example, successive commanders have widely varying personal styles to which the system must adapt: just consider the command styles of Commanders-in-Chief Eisenhower, Kennedy, Johnson, and Nixon. The official one reports to makes a big difference in how data are organized, how much of it one reports up the line, and the like. Another example is the unpredictability of one's own status of forces: the individual can't predict, for instance, when he might be told all Beechcrafts will be replaced with B-52s. This makes it very difficult to organize in advance how to process the information, and creates some of the problems that Dr. Bergman presents in his paper. Sometimes, if the environment is fairly predictable, functional flexibility can be traded for speed, many things can be put in hardware, and the system comes out way ahead. In cases where speed can not be traded for functional flexibility, it's not so easy to do that.

Another consideration is that the environment is hostile. NASA can count on the moon staying the moon and doing only what Nature does to try to outwit man. On the other hand, command and control systems have to consider that inputs may be spoofs; they have to worry much more about data security and people trying to penetrate the system to take advantage of it.

Finally, the stakes are not dollars as they are in commercial systems; they are not individual hazards as they are, primarily, in the Apollo mission; they are national survival. This means considerably more weight must be put on such things as software certification. If software says that the rising moon is a massive missile raid, the country may be in a lot of hot water.

The worst part is that all of these considerations interact with each other. An unpredictable environment requires a quick-change software capability. However, data have shown that quick-change software patching traditionally introduces many errors in the software; so, somehow or other, software must be organized not only to change it quickly but also to certify that the change is correct.

IMPLICATIONS FOR INFORMATION PROCESSING R&D

If one tries to infer from these considerations what is most needed in information processing R&D, the response may be

surprising. It is not voice recognition, nor image processing, nor large screen displays. Some fairly mundane things are required; for instance, getting the computer to help more in doing requirements analysis, in providing paradigms for developing and maintaining the system design, in exercising the system so there are no big mismatches between theoretical command and control performance and actual command and control performance, as often seems to crop up in things like the Pueblo incident, the Liberty incident, the EC-121 incident, and the like. Right now, the usual command and control system exercise is a very tedious, very manual kind of operation, which may take more than a year to prepare for, run, and analyze.

Another extremely critical R&D area involves software and system certification. Once again, the country is being bet on a correctly working software, and, typically, many bugs are found in the command and control software, as there are in everything else. This isn't intrinsic just to Air Force operations – on Apollo 14, fourteen software problems were found in a ten-day mission. It's just very difficult for technology to certify the correctness of software, but very important to do that.

Data security is another important R&D area. Many people are counting on having a data security box which allows multiple access to a common data base by all sorts of users on airbase loading docks, the commander's console, and practically everywhere else. The assumption is that nobody will be able to poke into unauthorized data, but the technology for guaranteeing that just isn't around.

INSTITUTIONAL PROBLEMS

In addition to technical problems in R&D support of command and control requirements, there are a number of institutional problems – like procurement policies – which never seem to track the pace of technology very well. Another problem is coupling the R&D that goes on in the Air Force with the operations. In some sense, this symposium is a symptom of this. For every man here who has to operate something in the Navy, there are about ten people in R&D organizations. This makes it very difficult to give the R&D community a good picture of what those operators really face in terms of day-to-day problems. It makes it very difficult for R&D people to convey to the operational people an appreciation of what advanced technology can give them.

A Critical Problem: Lack of Usage Data

Another major problem is that there is almost no data base on how computer systems are used within the Air Force (or elsewhere, for that matter). This lack creates several kinds of problems. For one thing, because what's going on is not known, R&D decisions

are made based on sample sizes of two or one or, fairly often, zero. Software is built without really knowing for what it's being built and it turns out to be unresponsive. Things are scheduled with little idea of the typical distribution of effort and one generally falls behind schedule and compromises to make up for it. As Dr. Bergman points out in Chapter 2, one often ends with inappropriate hardware assignments and very often tends to get a hardware view of the world.

The Compiler Development Learning Curve

Figure 4-1 illustrates the compiler development learning curve. It shows some relevant data on developing three successive FORTRAN compilers[1]. It shows that Effort 1 took 72 man-months; before checking the remaining data, the reader should try to guess what Efforts 2 and 3 took with each successive FORTRAN compiler. This answer is not, as one man who was supporting R&D in metacompilers claimed, that each new development of a compiler on a new machine takes about the same amount of time as the original job. Figure 4-1 indicates the opposite – that there can be a learning curve. There isn't always, necessarily, but there are things that can be done about it. This isn't to say that R&D in metacompilers should not be supported, but it is to say that many R&D decisions are being based on intuition that may not parallel the facts.

How Do Compilers Spend Their Time?

Figure 4-2 relates to unresponsive software; it contains some data taken by Knuth[2] in a study at Stanford on the distribution of complexity of FORTRAN statements. From the lefthand side of Fig. 4-2 the reader should try to guess what percentage of 100 typical FORTRAN statements were of the simple form A=B, how many had two operands on the right-hand side, etc. This should be of particular importance to a compiler designer because it would tell him how to optimize his compiler – whether it should do simple things well or whether it should do complex things well. The data in Fig. 4-2 show that 68 percent of these 250,000 statements were of the simple form A=B. When Knuth saw this and some similar distributions on the dimensionality of arrays, the length and nesting of DO loops, here was his reaction:

> "The author once found . . . great significance in the fact that a certain complicated method was able to translate the statement
>
> C(I*N+J):=((A+X)*y)+2.768((L-M)*(-K))/Z
>
> into only 19 machine instructions compared to the 21 instructions obtained by a previously published

EFFORT #1 :	72 MAN-MONTHS
EFFORT #2 :	36 MAN-MONTHS
EFFORT #3 :	14 MAN-MONTHS

Fig. 4-1. Compiler Development Learning Curve (McClure: FORTRAN Compilers, Successive Machines)

method. . . The fact that arithmetic expressions usually have an average length of only two operands, in practice, would have been a great shock to the author at that time!"

Thus, evidence indicates that batch compilers generally do very simple things and one should really be optimizing batch compilers to do simple things. This could be similarly the case with compilers and interpreters for online systems; however, nobody has collected the data for those, so it isn't known for sure, and people will continue to design compilers with nothing but fallible intuition as a guide.

Software Development Planning

Figure 4-3 relates to the problem of unrealistic schedules. Generally, a software effort is begun with a system analysis and, at some point, it is determined what the hardware and the software are supposed to do. From there, a schedule for software development is made up to interface with schedules for radars, training, operational procedures, and everything else. It would be very nice at that time to know what fraction of the effort is going to go into analysis and design, what fraction into coding and auditing, what fraction into program integration and testing. Figure 4-3 shows the actual results from some fairly big command and control and

COMPLEXITY		%
0	(A = B)	68
1	(A = B ⊕ C)	24
2	(A = B ⊕ C ⊕ D)	4
3		1
>3		3

Fig. 4-2. Complexity of FORTRAN Statements (Knuth Study: 440 Lockheed Programs; 250,000 Statements)

space-type projects[3]. And, as the reader can see, coding and auditing (auditing is basically desk-checking) doesn't take a lot of the total effort. The bottom line is Fred Brooks's estimate for OS/360[4]: 33 percent analysis and design; 17 percent coding and auditing; 50 percent testing. Often, people use the analysis and coding part of the program to make up the whole schedule. As a result, the schedule is written when they are near the end of their established schedule and suddenly there is still 50 percent of the job left to do. This causes some horrible compromises or it causes things to go out in the field inappropriately tested.

Hardware/Software Tradeoffs

Figure 4-4 complements what Dr. Bergman mentions about Parkinson's Law. As the drive increases toward complete use of the speed and memory capacities of hardware, what happens to software costs? Do they stay relatively constant or do they start increasing slightly? As Fig. 4-4 shows, they escalate asymptotically as 100 percent hardware utilization[5] is approached. This should be very important to how the initial sizing of hardware is done. Typically, however, what people do is size the job, add about 15 percent for growth or uncertainty, buy the hardware, and then set software personnel to work. That means that they are about at the 85 percent utilization point on the curve and that means almost doubling relative programming costs. Further analysis indicates that hardware should be overbought by 50 to 100 percent to minimize

	ANALYSIS AND DESIGN	CODING AND AUDITING	CHECKOUT AND TEST
SAGE	39%	14%	47%
NTDS	30	20	50
GEMINI	36	17	47
SATURN V	32	24	44
OS/360	33	17	50

Fig. 4-3. Computer Program Development Breakdown

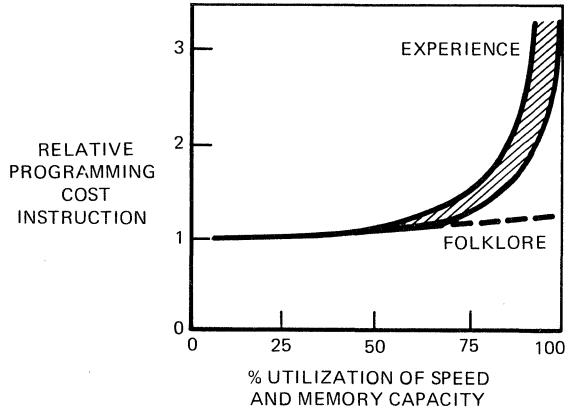


Fig. 4-4. On-Board Computing: Software Costs

total hardware-software costs – if the Parkinsonian tendency to fill up the added capacity with marginally useful tasks[6] can be avoided.

Computer/User Tradeoffs

Figures 4-5, 4-6, and 4-7 concern overconcentration on machine aspects. Some time ago an experiment was done to see how certain kinds of response characteristics in a time-sharing system affected the way people solved problems. In particular, the "lock-out" period was varied. Suppose the individual is solving a problem at a console. He thinks of something he wants to do. He

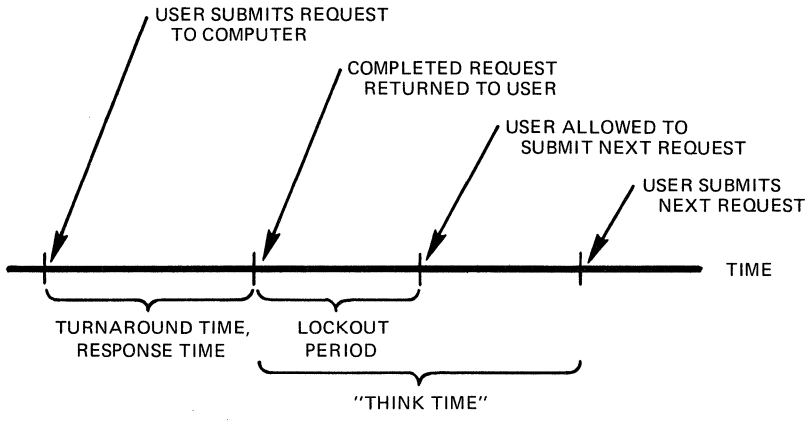


Fig. 4-5. Sequence of Events for Submitting a Trial Solution

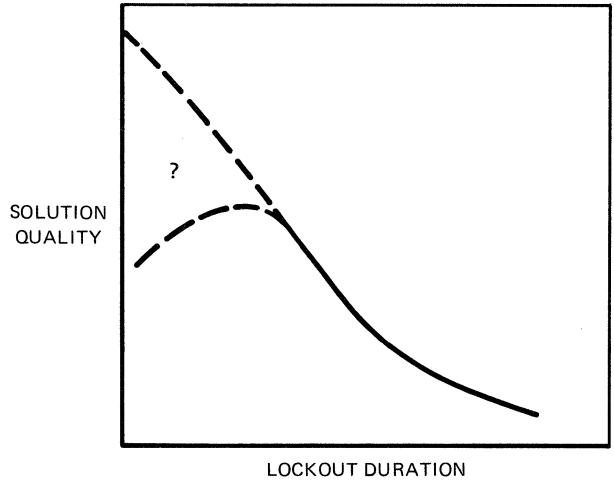


Fig. 4-6. Problem Solving Efficiency (Theoretical)

puts some instructions in and he hits the equivalent of the "go" button. The machine grinds around for a while, and, in a certain amount of time called the "turnaround time" returns the completed request to the user. At that time, the machine might tell him, "I'm

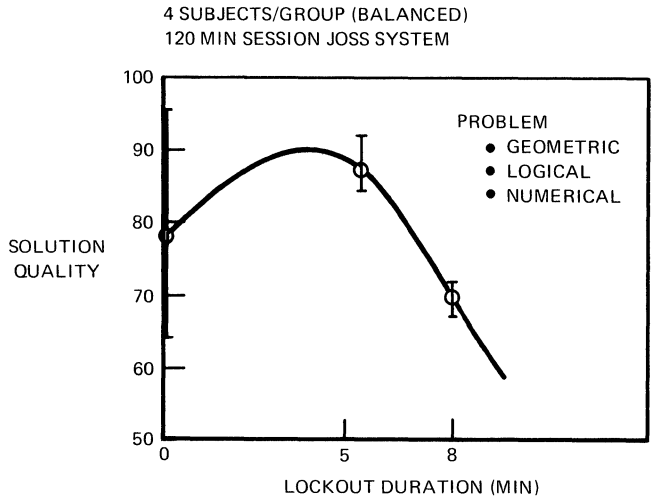


Fig. 4-7. Problem Solving Efficiency (Actual)

going to be busy with other things for five minutes, therefore you're effectively locked out for this time," or it might give him immediate access to the machine for his next request. The resulting delay, if any, is called "lockout time." The time between receiving his request and submitting the next one is typically called "think time." Figure 4-5 illustrates the relationships between the various time periods.

For an infinite lockout time, very poor performance would be expected, but it's not completely clear that if one went to a zero lockout time and gave the user immediate access to the computer, he would necessarily do better (Fig. 4-6). As a matter of fact, before this experiment was done, there was a hypothesis by Gold[7] at MIT which indicated that immediate access to the computer might get people to concentrate on the tactics of problem-solving rather than the strategy, and that, by imposing a lockout period, a floor might be put under the user's think time and make him think more. And, as a matter of fact, that's what happened in this example. This was a problem in a geographic area servicing problem; 20 graduate students were asked to locate three hospitals on a grid map of a city in such a way as to minimize the response time to emergencies in that city. And, as the reader can see, locking them out for five minutes produced better performance than giving them immediate access to the computer[8].

One implication of this is that concentrating on just the hardware aspects of computing systems will not necessarily provide the optimal solution for people's performance. And there are a lot of tradeoffs being made nowadays – large blocking of computer input and output, memory residence limitations, restricted debugging options – that increase machine efficiency but get in the

user's way of solving problems. These tradeoffs should be looked at very carefully. Once again, though, hardly anybody collects relevant data.

A FUNDAMENTAL PROBLEM: CONFLICTING WORLD VIEWS

Figure 4-8 illustrates the points made by Jim Burrows that some features can look good to system designers and not so good to users. During the past few years, I have had alternately the roles of a graphic system developer, viewer of demonstrations (a role which is often the key to continued R&D funding), interactor with production users like command and control people, and research user of graphic systems. Through an informal survey, I found that system developers are not necessarily sensitive to the things that users are sensitive to; Fig. 4-8 lists some graphic system characteristics and associated sensitivities.

Are displays properly centered and balanced? This is something that the system developer generally worries a lot about. It looks good when someone comes in for a demonstration. A production user or research user doesn't care much whether the display is centered or not. Similarly, if scrolling is jerky rather than smooth, the system developer worries about that; the demonstration viewer is impressed if it's smooth; the users don't really care that

SOME GRAPHIC SYSTEM SENSITIVITIES	SYSTEM DEVELOPERS	DEMO. VIEWERS	PRODUCTION USERS	RESEARCH USERS
ARE DISPLAYS PROPERLY CENTERED AND BALANCED?	HIGH	HIGH	LOW–MED	LOW
IS SCROLLING SMOOTH RATHER THAN JERKY?	HIGH	HIGH	LOW–MED	LOW
DOES PROGRAM HAVE "HELP" PAGES?	LOW–MED	HIGH	HIGH	MED–HIGH
IS THERE MORE THAN ONE WAY TO DO THE SAME THING?	MED	LOW	HIGH	HIGH
WILL THE SYSTEM BE OPERATIONAL TOMORROW?	LOW–MED	LOW	HIGH	HIGH
ARE PROCEDURES CONSTANT FROM WEEK TO WEEK?	LOW–MED	LOW	HIGH	MED–HIGH
CAN A NEW USER-OPTION BE ADDED BY NEXT WEEK?	MED–HIGH	LOW	MED	HIGH
DOES PROGRAM INVOLVE SOPHISTICATED ALGORITHMS?	HIGH	MED	LOW	LOW–MED
DOES PROGRAM MEASURE USER CHARACTERISTICS?	LOW–MED	LOW	LOW	LOW

Fig. 4-8. Graphic System Characteristics and Associated Developer/User Sensitivities

much. Does the program have "help" pages – does it tell him what to do if he makes a mistake? Or does it try to compensate for the mistake? The system developer doesn't need that kind of thing, so he doesn't care much about it. It looks good to a demonstration viewer because he generally doesn't know that much about the system, and it's very useful for production users. Some research users get used to the system and learn to compensate and it's not quite that important.

Is there more than one way to do the same thing? Again, this is something the system developer occasionally finds useful. The demonstration viewer is there only one time and he doesn't really appreciate that kind of thing, but in the production or research user business, it's often very helpful to have alternative ways of specifying the same thing. Sometimes one piece of hardware may be down or he may be using his other hand for something else.

Similarly for these questions: Will the system be operational tomorrow? Are procedures constant from week to week? Can a new user option be added by next week? The system developer isn't that concerned about most of these, particularly if the development job is considered a research project. The demonstration viewer isn't going to be there tomorrow or next week, so he has very low sensitivity to this. But the research and production users are very sensitive to these aspects.

Does the program involve sophisticated algorithms? Does it have to get deep into graph theory or list processing to do the job? The system developer often has a high sensitivity to this. If he can structure the problem so that it does, that's so many more papers that he can prepare for conferences and journals. He can also be more impressive when he talks to the demonstration viewer and that makes the demonstration viewer more impressed by the system. The users really don't care that much. If the job can be done by brute force at a little bit more cost in efficiency but maybe a little bit more in understandability and program maintainability, they will prefer it that way.

Last and worst of all is the question – does the program measure user characteristics? For some reason or other, most system developers that I've seen aren't really concerned that much about what the distribution of response times are, what the distribution of errors are when people use the systems, or how these things vary with the kind of users – military, researcher, or graduate student. And, unfortunately, the demonstration viewers and the users really don't care that much either.

IS COMPUTER SCIENCE CURRENTLY A SCIENCE?

The phenomenon of nobody really being that concerned with measuring what's going on prevents computer science from approaching a true science. My definition of computer science paraphrases a letter to Science magazine written by Newell, Perlis,

and Simon[9]: Computer science is the application of scientific method to phenomena involving computers.

Now, what is scientific method? Scientific method first involves the systematic collection of observations. To build a new interactive system requires making a fairly systematic approach to potential users, potential developers, and potential maintainers of the system; finding out what characteristics of the new system are likely to be most important and less important for each group concerned, and gathering as much data as possible on relevant similar systems. Next, some hypotheses would be constructed saying the following additional tools or additional techniques would help in providing this set of system characteristics. Then, critical experiments would be designed and performed. A system would be built and tried out on people to see whether its usage characteristics really did verify the hypotheses. Thus, the resulting data would be analyzed and iterated around, possibly generating new hypotheses from this further systematic collection of observations.

However, in computer science R&D, there is usually a very nonsystematic collecting of observations. The "early Knuths" observed that they had a great deal of complexity in their own computing programs, and therefore built compilers that were usually optimized around compiling complex expressions. Those are the kind of hypotheses that are constructed. There is almost no design and performance of critical experiments to validate those hypotheses. Too often the next iteration consists of the developer saying, "I just heard about the following list-processing technique or garbage-collection technique and I'm sure that will make my next compiler more efficient, or my next online system more efficient, " and he proceeds to the next R&D project with little more than that as a basis.

The university disciplines that this pattern matches are not what are generally considered as science, like physics, or biology. What it seems closest to is basket weaving. A basket weaver has a very difficult job. He must plan his basket very carefully and he puts a lot of loving care into it; he builds it, studies it from various angles, discusses it with other basket weavers, and then goes off to build another basket. Very rarely though does he go out and sample users to find out whether they are interested in baskets with handles or with several compartments rather than one compartment, and the like. And, unless something changes considerably in computing, it will remain a kind of computer basket weaving.

What Can Be Done to Make Computing a Science?

The scientific method here provides a good paradigm for judging research proposals and research products to see whether, indeed, they are based on a systematic collection of observations, whether they have a careful experimental design, followed by the collection and analysis of data to see whether it actually verifies

the hypotheses put forward. I would hope that in years to come, as we do try to make computing a science, that more R&D will be done this way.

REFERENCES

1. McClure, R. M., "Projection Vs. Performance in Software Production," in Naur, P. and Randell, B. (Ed.), Software Engineering, NATO Conference Report, January 1969.

2. Knuth, D. E., "An Empirical Study of Computer Programs," Stanford Computer Science Department, Report CS-186, 1971.

3. Manus, S. D., "SDC Recommendations for Spaceborne Software Management," Proceedings, First Spaceborne Computer Software Workshop, 1966, pp. 345-360.

4. Personal Communication from Professor F. P. Brooks, University of North Carolina, 1970.

5. Williman, A. O. and C. O'Donnell, "Through the Central 'Multiprocessor' Avionics Enters the Computer Era," Astronautics & Aeronautics, July 1970.

6. Boehm, B. W., "Some Information Processing Implications of Air Force Space Missions in the 1970s," Astronautics & Aeronautics, January 1971, pp. 42-50.

7. Gold, M. M., "Methodology for Evaluating Time-Shared Computer Usage," Ph. D. Dissertation, Massachusetts Institute of Technology, 1967.

8. Boehm, B. W., M. J. Seven, and R. A. Watson, "Interactive Problem-Solving – An Experimental Study of 'Lockout' Effects," Proceedings, Spring Joint Computer Conference, Vol. 38, 1971, pp. 205-210.

9. Newell, A., A. J. Perlis, and H. A. Simon, "Computer Science," Science, Vol. 157, September 1967, pp. 1373-1374.

5. Scientific Analysis Computational Requirements

Ralph H. Pennington

System Development Corporation

Santa Monica, California

The areas involved in scientific analysis are so broad that I don't really know how to make a thorough survey of them. I thought I would start by describing how a particular kind of scientific program typically gets done on a computer and try to extrapolate from that both to requirements for computer capacity for some classes of scientific calculations and to requirements for support to the programmer in putting together such computations.

For at least a large class of scientific calculations, the starting point is some set of differential equations or partial differential equations that supposedly describe the physical laws that govern some system behavior. To find the solution to a problem, the steps are to make whatever simplifying assumptions one can – to get those laws expressed in as simple a form as possible – and then to replace the partial derivatives by difference equations.

One example of the above is a calculation in one-dimensional hydrodynamics. The problem is a medium which is uniform in all directions but one (Fig. 5-1). Typically, a shockwave or some motion is occurring in that direction (X) so that a point along the line of motion will tend to move under the influence of some force. A shockwave may cause this point to move to the right and then back to the left. To find out about that motion, the point for discussion must be identified and then the motion computed as a function of time. To do the computation involves keeping track of the velocity with which that point moves, the pressure that is causing it to move, the internal energy in the little region about the point, and either the density or the specific volume of the material around that point. Of the basic equations of physics that are involved the first one is simply Newton's Law: F = ma. The first differential equation in Fig. 5-1 states this law backwards, ma = F. The force is the change of pressure with respect to distance, derivative of P with respect to x; the mass is essentially the density

EQUATION OF MOTION (F = ma)

$$\rho o \frac{\partial U}{\partial t} = - \frac{\partial P}{\partial x}$$

CONSERVATION OF ENERGY

$$\frac{\partial E}{\partial t} + P \frac{\partial V}{\partial t} = 0$$

CONSERVATION OF MATTER

$$\rho o \frac{\partial V}{\partial t} = \frac{\partial U}{\partial x}$$

EQUATION OF STATE

$$P = P (\rho, E)$$

X = COORDINATE OF POINT

U = VELOCITY OF POINT $(= \frac{dx}{dt})$

P = PRESSURE

E = INTERNAL ENERGY

V = SPECIFIC VOLUME $(= \frac{1}{\rho})$

ρ = DENSITY

Fig. 5-1. Example One-Dimensional Hydrodynamics

in this coordinate system; and the acceleration is the change in velocity with respect to time. The equation that describes the conservation of energy and one that describes the conservation of matter are also used. Then, typically in physical systems, some other equation describes the relation of the pressure in the medium to the density and the energy in that medium. The simplest form that most individuals learned in college physics is Charles's Law or Boyle's Law or something like that. For the more elaborate systems, physicists in laboratories seek the equations of state for particular materials under particular conditions and determine empirical formulas to represent them.

Solving this set of differential equations requires turning them into difference equations; this is done by splitting the material into regions or zones and considering the properties inside any particular zone to be constant (or at least have a preselected variation about some average value). In Fig. 5-2, small arrows mark the boundaries of the zones. If the zones are numbered from left to right – one, two, three, and so forth – zone J is being discussed here. One of the first problems in making a difference equation out of this is where the values of pressure, velocity, etc., should be considered to apply. I've shown in this case that pressure is

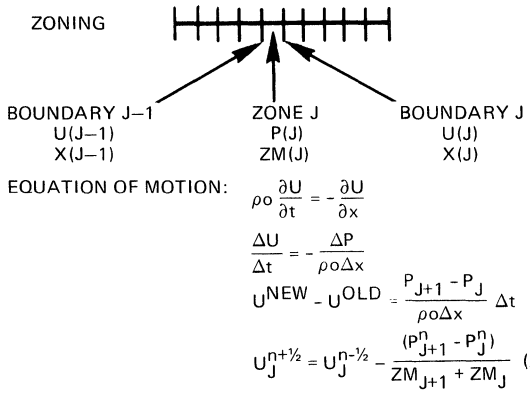


Fig. 5-2. One Dimensional Hydrodynamics Typical Difference Equations

representing a value at the center of a zone and have also shown the mass (representing the density) to be carried at the center of the zone. The coordinates X(J) and X(J-1) are the coordinates of the boundary of the Jth zone, and the values of velocity U(J) and U(J-1) are measured at the boundary of the zone rather than the center of the zone. This suggests, right away, the first problem in getting from the differential equation to the difference equation. That is, there are a number of options on how to make the selection of points at which the values of pressure, velocity, etc., are assumed to apply.

The equation of motion in Fig. 5-2 is converted from a differential equation to a difference equation in a series of steps. In the first step, the derivative symbols are replaced by Δ's and the ρ_0 is moved to the other side of the equation. Each of the Δ's supposedly represents the difference of value between two points. Now in the $\Delta U/\Delta t$, the Δ stands for change of U in time, so it is some value of U at a new time minus a value of U at an old time. The ΔP stands for a change in pressure for some Δx, so that it is the P at some zone minus the P at some other zone. I have elected to use P(J+1) - P(J). Typically in the figure, the subscripts represent the X position of zone boundaries and superscripts represent time. The difference equation finally takes a form that says the U(J) at some new time n+1/2 is equal to U(J) at time n-1/2, minus some terms that involve Δt's at various times. This is the form

in which the equation is typically used in actual computations. It seems somewhat odd to write the difference equation in this form. Why wasn't U employed at time n+1 and U at time n, rather than U at time n+1/2 and U at time n-1/2? The answer is that the method won't work done that way. If this appears to be a black art, that's what it is. This sort of situation has always been rather embarrassing to applied mathematicians. In constructing a difference equation to represent a differential equation, there are two basic problems to worry about. One is stability. That difference equation has to be such that as one steps through time using that equation the solution doesn't wander off from the solution for the differential equation. The other problem is one of convergence. If the Δt is made smaller and smaller and the problem is rerun, answers should get closer and closer to the differential equation.

There are nice mathematical theorems about convergence and stability, but when it comes down to writing differencing schemes that will be stable and will converge, I am afraid that the mathematical theorems haven't been much help. The physicists have discovered the successful forms for differencing schemes by trial and error (I think they call it "physical intuition").

The process reviewed here for one simple equation in Fig. 5-2 has been repeated countless times for countless versions of physical problems. No neat, simple system has emerged that works in all cases. What may be a convergent scheme in one set of circumstances will tend to be a divergent in another set of circumstances. Schemes known to be unconditionally convergent turn out to be very poor computationally (i.e., require inordinate amounts of computer time or give poor accuracy). Experimentation with differencing schemes will be a continuing area for computer application, one where interactive capabilities can be of great assistance.

Now, the next consideration is the amount of computation required to solve the equations once a suitable difference scheme is found. That difference equation in Fig. 5-2 tells what must be done to get U at the next time, given the value at the preceding time. One pressure must be subtracted from another, which is one mathematical operation. Two Δ's must be added, which is another one. Next, there is the addition of two zone masses, a multiplication and a division, and then another subtraction. So a half a dozen mathematical operations are required to do that one step for that one equation.

What does a full calculation of that type mean in terms of computer requirements? In the total set of equations in Fig. 5-1 were five variables in the system that would have to be updated from one time step to the next. There was position, velocity, pressure, energy, and density (see Fig. 5-3). What has been labeled V in Fig. 5-3 is really the reciprocal of the density, usually called specific volume. Typically, the number of zones

NR. VARIABLES = 5(X, U, P, E, V)
NR. ZONES = 1000
OPERATIONS/ZONE/TIME STEP ≈ 30
NR. TIME STEPS ≈ 1000
TOTAL STORAGE 5000 WORDS
TOTAL OPERATIONS 30 MILLION

Fig. 5-3. Computer Requirements One-Dimensional Hydrodynamics

required to track a shockwave through a medium is on the order of a thousand. The equation for moving one variable in one zone from one time step to the next requires half a dozen operations. So one may assume that 30 or so operations will be necessary to get all five variables at one zone point updated from one point in time to the next. The number of time steps required to step from the initial time to the end of the time of interest is determined by a few hidden rules. For example, in this kind of problem there is a so-called Courant condition that says the time step has to be small enough that the sound waves cannot travel more than one zone in one time step. This means that it will require at least 1000 time steps for a wave to get all the way across the thousand zones. So this might be a problem of interest; five variables, 1000 zones, 30 operations per zone per time step, 1000 time steps. A problem of this size takes a reasonably nominal amount of storage – 5000 words – to store these five variables for each of the thousand zones. There are 1000 zones – 30 operations per time step – or 30,000 operations and there are 1000 time steps, so 30 million operations are needed to get through the problem. Again, not a terribly large number for the faster machines available today. A machine such as the CDC 6600, for example, probably does two or three million instructions per second. Any computer program includes a number of things besides arithmetic operations (for example, fetches from and stores to memory, program branching, etc.) so that the two or three million instructions per second might correspond to perhaps one million arithmetic operations per second. Thus on a machine of the 6600 class approximately 30 seconds or so are required for this type of problem.

Now, what happens if one more space dimension is added to the hydrodynamics problem and the same sort of computation is done? The extrapolation is made as simple as possible. If there are two dimensions, the problem variables will obviously have to include both an X and a Y instead of just an X. A velocity is needed in the X and Y directions instead of just in an X direction, so at least seven problem variables exist instead of five in the

two dimensional case. There will be 1000 zones in the X direction and 1000 zones in the Y direction, giving a total of a million zones.

The following is a simple estimate of the number of operations involved. If the same sorts of difference equations could be used as in the one-dimensional case, then whatever is done in X direction is also done in the Y direction, so there will be twice as many to do at each point. The same number of steps will be done.

Figure 5-4 summarizes the situation for the two dimensional case, given the above assumptions. Seven million words of storage and 60 billion arithmetic operations now are needed, or about 16 hours on a machine that could do a million operations per second. This is a little far from reality. Typically, two-dimensional problems use more like 100 zones in each direction, rather than 1000. Such problems are indeed run for practical purposes these days. Even with 100 zones in each direction, they usually require several tens of hours on a 6000 class machine, because the actual difference equations tend to be much more complex than the ones demonstrated above and to require many more operations per time step.

It appears that a two-dimensional hydrodynamics problem pretty well saturates the largest of the machines generally available today. Yet, from the standpoint of the physics, this type of problem is only the threshold of all sorts of classes of computations one would like to do. For example, back in the one-dimensional case – if the internal energy gets high enough in some zones, radiation appears. One has to worry about the redistribution of energy by radiation diffusion or, if the temperature gets even higher, one has to worry about radiation transport. When equations for radiation transport are put into the one-dimensional hydrodynamics problem, it essentially becomes as big as a two-dimensional problem with hydrodynamics only.

Ten years ago, when the fastest machine was the IBM 7094 Model 2, a one-dimensional hydrodynamics was just the biggest problem anyone did. Now the current generation of machines, the CDC 6600 and that class, have made two-dimensional hydrodynamic problems standard. Such problems are done in production form. The next set of machines now coming along, the IBM 195 and the CDC 7600, are not powerful enough to allow one to take the next step – to do three-dimensional hydrodynamics problems, for example. That step has a multiplying factor both in storage and in arithmetic operations similar to the step from one-dimension to two. A mere factor of five in computer speed isn't enough.

There are a number of interesting two-dimensional hydrodynamics problems that can't be done in a practical way on current computer equipment; for example, two-dimensional hydrodynamics involving radiation transport or two-dimensional hydrodynamics with chemical kinetics. There are many classes of physical problems of this sort which can't yet be attacked simply because of machine size although the methods of computer solution are known and understood.

	ONE DIMENSION	TWO DIMENSIONS
NR. VARIABLES	5	7
NR. ZONES	1000	10^6
OPS/ZONE/TIME STEP	30	60
NR. TIME STEPS	1000	1000
TOTAL STORAGE	5000 WORDS	7 MILLION WORDS
TOTAL OPERATIONS	30 MILLION	60 BILLION
	(30 SECONDS ON 1 MOPS MACHINE)	(~16 HOURS ON 1 MOPS MACHINE)

Fig. 5-4. Computer Requirements Extension to Two Dimensions

There is a question of how big a computation is worth trying to do on a particular problem. For example, following a missile flight can be done at different levels of detail with different computational requirements as indicated in Fig. 5-5. The missile can be considered as a point mass and Newton's Laws applied and that equation integrated with some very small number of operations, perhaps 10^4 or so. To be more elaborate, a six-degrees-of-freedom calculation can be performed. This kind of thing is done on the more detailed trajectory calculations these days where drag effects in the atmosphere can be considered. One may even put in such things as variations of gravity with position, variation of atmosphere with position, and rotating earth (if an earth reentry calculation is being done). Such calculations typically may require on the order of 10^7 total operations. Here again, on the machine that can do close to a million operations per second that turns out to be not much of a problem. One could conceive of trying to do such a problem in three-dimensional hydrodynamics. To do even one position in space in three-dimensional hydrodynamics would require on the order of 10^{12} operations. To track an entire trajectory would require a few orders of magnitude added on the top. The problem has gone from one that is easily tractable on a current machine to one for which another generation or two of computers will be needed to even hope to begin. Is it worth even considering such calculations? Why should one be interested? Certainly a six-degree-to-the-freedom calculation, so far as the trajectory behavior is concerned, gives one essentially everything he wants to know.

THREE DEGREE OF FREEDOM	10^4 OPERATIONS
SIX DEGREE OF FREEDOM	10^7 OPERATIONS
THREE DIMENSIONAL HYDRODYNAMICS	10^{12} OPERATIONS

Fig. 5-5. Spectrum of Computation Sizes Example: Missile Flight

There are several reasons for wanting to do calculations of the larger classes that are beyond the sizes that can currently be done.

One reason is that for any detailed data about what's going on the calculation is probably cheaper than any experiment. A two-dimensional hydrodynamics calculation might take 16 hours or so on a very fast machine. That might cost $1000 an hour or so to run and that's $16,000 for one pseudoexperiment on that machine. For that price, there are very few real life experiments that allow taking detailed measurements about hydrodynamic flow in a system environment.

Another reason is that calculations do provide more data than an experiment. To do some experiment to find flow patterns about missiles or to do a detonation and try to find the way that the explosive expands, the instrumentation problem is extremely difficult and only a very few points can be instrumented and usually not enough points to form a complete picture of what is going on. One of the main things a computation may do is give enough understanding of what the data points really mean by providing

- CHEAPER THAN EXPERIMENTS
- PROVIDE MORE DATA THAN EXPERIMENTS
- NEEDED TO INTERPRET EXPERIMENTS
- NEEDED WHERE EXPERIMENTS INFEASIBLE

Fig. 5-6. Reasons for Large-Scale Computations

computational data points throughout the regions where experimental points weren't obtained.

The third point is that one needs rather elaborate computations in many cases to interpret experiments correctly. I guess my favorite whipping boy in this area is air chemistry. Typically, in setting up an experiment one uses some sort of evacuation chamber, lets the right elements into the chamber, and tries to determine by observation the rate at which the reaction he is interested in occurs in that chamber. He does this by trying to design the experiment very carefully, so that the only reaction that takes place is the one that he is interested in. The literature over the past many years is full of instances where somebody came out with a cross-section for recombination of O_2^+ or something like this and three years later someone repeated the experiment and said, "No, the cross-section is wrong by two orders of magnitude because the O_2^+ reaction was dominated by an O_4 reaction." To interpret an experiment of that type correctly, one needs to have a mathematical representation which does not assume out of existence all of the effects except the one the experimenter is looking for, but instead allows all of the presumed second-order effects to be represented and computed so that he can get a real feel for whether they are affecting the experiment. There are cases, of course, where experiments are absolutely infeasible. One that plagued the nuclear people for some time back, when it was appropriate to be testing, was to get detailed measurements very early inside of a fireball where the environment is almost impossible for any instrumentation. Such situations may be forced to rely on computer data. Then the problem is to have the computation realistic enough so that one can use subsidiary experiments and an elaborate computation to allow one to tie the specific physical behavior to very indirect experimental observations.

Some of the applications areas where there would be definite value in doing some of the very large-scale computations are:

- MISSILE FLIGHT SNAPSHOTS
- RE-ENTRY WAKE PHYSICS
- CONVENTIONAL DETONATION HYDRODYNAMICS
- INSTRUMENTATION PHYSICS
- PHASED ARRAY RADAR RADIATION FIELDS
- UNDERWATER SIGNAL PROPAGATION

Fig. 5-7. Application Areas Needing Large-Scale Computations

I think that in each of these areas there is some experimental data, there is a general conceptual understanding of what goes on in the area, but there is little in the way of detailed modeling that allows one to really understand, in intimate detail, what is happening physically. The applications areas listed are ones in which there were fairly definite disciplines for doing calculations before the advent of computers. Many techniques for doing the large-scale hydrodynamics and radiation transport calculations did grow up with the nuclear community because computers came along about the same time the problem came along. There are many, many areas where classical methods of hydrodynamics or of electromagnetic series are used because the disciplines grew up before the age of computers. A set of techniques which were as good as one could get in that environment also grew up. Computers are only now beginning to be applied in those areas. I think there will be a gradual awakening to the fact that careful numerical solutions can frequently eliminate the need for the gross simplifying assumptions so often required to obtain classical solutions. So I think the next decade will continue to see a recognition of the need for work on very-large-scale computations in a widening arena of applications areas.

I have mostly considered the requirements for very, very large computations. However, I began with a sample of a rather small computation and even at that level there undoubtedly will be continuing requirements for further repetitions of such computations with different environments, with different materials, with different equations to state, and so forth. Throughout the spectrum there are a certain number of requirements associated with the use of computers in the area of interactive use for general computational purposes. Some of these requirements are shown in Fig. 5-8.

I think most have found it very nice to be able to develop small code modules on a time-share system and do an initial experimentation on such a system. I think that over the past several years, most have found that the next step is always a terrible one. When one gets a few pieces of code put together on a time-share system, understands how they work, and wants to build his big system, he usually is stuck with the problem that he can't even extract the code from the system, get it punched out on a deck, take it over to a big machine, and put it in without several administrative steps in between. Ideally, one would like to debug his small code modules online, put together a sufficient library management system, and edit these small code modules into a much larger package of code. At that point, he probably stops caring very much about whether he can operate interactively or not. He is willing to submit the problem to be run on the batch basis to a large machine and wait for his outputs.

The problems of handling outputs can become quite difficult. One thing that kept two-dimensional hydrodynamics codes from

- DEVELOPMENT AND DEBUG OF SMALL CODE MODULES
- SCAN AND EDIT OF SOURCE CODE LIBRARIES
- MERGE OF CODE MODULES AND SUBMITTAL FOR BATCH RUN
- SCAN AND ANALYSIS OF BATCH RUN OUTPUTS
- GRACEFUL GROWTH OF PROGRAMS

Fig. 5-8. Interactive Requirements for Scientific Use of Computers

working for a number of years was that it was impossible to look at the data and debug the code. If one tried to print out the values – pressure, energy, density, and so forth – for every zone on a 100 x 100 grid, and then sat there looking at those numbers and trying to figure out what happened during the machine run, he just couldn't quite make it. The full debugging of such programs waited until there were enough plot routines that, at every step in the program where he wanted to, one could look at two-dimensional contour plots of the pressure and energy and density and things like that. The plots ended being the chief debugging aids. The two-dimensional contour plots took about as much machine time as the computation itself, but the time turned out to be well spent from the standpoint of getting codes debugged and useful results generated.

Once one has performed some significant number of large-scale computations, an adequate storage and retrieval system for the results, preferably in graphical display form, can reduce the need for additional large-scale computations. At any one time, an individual usually is interested only in a few pieces of information that result from some large-scale calculation. However, at a later date he or others may be interested in other results from the same calculation. A suitable library of stored results and convenient retrieval methods provide such data without the requirement for rerun.

I have discussed large-scale computations at length and, in the process, have perhaps slighted the interactive use of computers for smaller scale scientific work. While the interactive

solution of small problems is an important area, I have tried to emphasize where I believe the priorities should be.

6. Process Control System Requirements

Peter Swerling

Technology Service Corporation

Santa Monica, California

The focus of this chapter is somewhat different from that of many of the others, nevertheless it is no less significant in terms of applications of multi-access interactive computing technology for the 1970s. Certainly, the kinds of applications presented here do not take a hind seat insofar as the requirements placed on computer technology in this period are concerned.

Basically, the subject matter of this chapter differs in that it discusses interactive systems involving machine-machine interactions. In other words, the users are not necessarily human beings, but other machines or, perhaps, even elements or subelements of machines. For example, the users may be a multiplicity of sensing devices such as radars, or they may be missiles which are being guided under control of a computer, or they may be subelements of individual sensors. It's also possible for the users to be thought of as functions, rather than as specific hardware items – for example, different sensor functions such as search, track, guidance of an interceptor, or discrimination.

Now, despite this somewhat different focus, it will be apparent from the examples that the systems presented are definitely full-fledged, multi-access, interactive systems. They preserve all of the essential features of such systems. The users, here considered in the sense just mentioned, compete for computer time. In fact, a major problem in the use of such systems, which is already recognized today, is the allocation, or management, of computer resources. They preserve the feature of interactivity. For example, the computer may process information input from a large multiplicity of sensors or sensor elements and also control the subsequent user activity. In addition, there is often the necessity to respond to unpredictable stimuli or at least not wholly predictable stimuli.

An example of the specific applications now becoming increasingly important is the ballistic missile defense battle

management problem. A very closely related problem is that of the ground-to-air or sea-to-air defenses against attacking forces which are not ballistic missiles but which are perhaps aircraft or shorter-range missiles. These systems, as they are currently developing, embody a multiplicity of sensors, mostly radars in this case; a multiplicity of weapons to be committed against the attacking force; and one or more computers of different levels, perhaps ranging from very specialized hard-wired computers to big general-purpose computers. The computers process the information from the sensors and, in turn, control not only the commitment of weapons to the attack force and the guidance of those weapons but also what the sensors do from moment to moment – that is, where the radars should be pointing their beams, what type of wave forms they should be transmitting, and so forth.

Incidentally, the above is not meant to convey a Strangelovian picture of the national fate being controlled by machines talking to other machines. Obviously, the human being would intervene at least in the decision of whether the battle was going to start, but the events are taking place at a sufficiently rapid rate that most of the functions mentioned have to be automated and conducted under the control of a computer or a set of computers.

The same type of machine-machine interactive and multiple access system may have nonmilitary applications. The control of traffic lights in a city is one example. The sensors or "users," as far as the input sensing elements would be concerned, might be elements which sense the traffic flow over given streets. The computer would process these and decide according to some decision algorithm how to control the traffic lights.

Another application, which will be emphasized later, is the satellite communications problem; that is, the use of satellites as communications relays, especially with regard to the so-called multi-access problem. That problem, of course, is squarely within the domain of the classical C^3, or command-control-communications problems in the military, but also has applications to nonmilitary government and commercial systems.

A number of these cases or specific examples may involve multi-access systems in which the access actually can be time-sequenced in a programmable manner so that they would involve what might be called time-programmable multi-access rather than random-time multi-access. On the other hand, when the system has to respond and allocate computer resources in a manner which is not completely predictable, it is a true random-time multi-access kind of case.

So far, some of the systems applications have been described. Another major point is that there is another level of multi-access, interactive systems which actually involve techniques applicable to the operation of just a single sensor. I think one of the great technological developments in the next 10 or 15 years, in the improvement of performance of individual sensors, will involve

the application of this kind of technology. As the reader will see, it is really regarding an individual sensor as a multi-access system of users.

Radar is an example. In the last 15 or so years, one of the big technical breakthroughs in the design of radar systems was the electronically scanned antenna. Electronically scanned antenna structures consist not of a typical dish but of an array of elements, each one of which is, at least potentially, separately excited or controlled. Up to this point, the array-of-elements structure in typical radars has been used to provide inertialess scanning; that is, very rapid switching of the radar beam from one point to another in space. But actually this represents only a partial use of the potentialities of that device. The full potentialities of that device will not be used until one implements a radar in which every single element in the array is regarded as a separate channel for incoming information with a separate receiver behind it and every single element is regarded as something which can be separately excited and whose activity can be separately controlled. Assuming the implementation problems can be solved, very great performance improvements can be achieved, at least in principle. These improvements can be achieved by going to what I like to call the "fully cybernetic radar"; that is, a radar which is an array structure of elements in which each element is regarded as a separate receiving channel for signals and each element is regarded as having an activity which can be separately controlled.

In this kind of case, there is an individual sensor in which every antenna element is a user so to speak, every individual antenna element separately puts signals into a computer and is separately controlled by a computer. The number of elements typically used in large radars ranges from hundreds up to thousands, so, in this case, there is a system under the control of a computer in which there are thousands of users. Even worse, the rate at which such users have to be programmed and controlled is very, very rapid. In other words, one may be talking about different excitations of the elements in periods of the order of microseconds and so these are not only very, very many users, but there must be very, very rapid control. In addition to the antenna elements, other aspects of the radar can also be regarded as degrees of freedom and, hence, as users of the multi-access system; for example, the specific waveforms which are transmitted. Generally, radars increasingly use not one individual transmitted waveform but so-called suites or collections of waveforms. These also have to be adaptively controlled in response to the specific environment in which the radar is operating at the moment, so those degrees of freedom would multiply the degrees of freedom involved in the individual antenna elements. In addition, sometimes it's desired to process the information in very many ways simultaneously. As many as a hundred or a thousand different simultaneous ways of processing information can be

easily a practical case. The simplest example of this would be in forming multiple beams simultaneously. Multiple beams formed simultaneously in a sensor on the receive end of the surveillance link amount essentially to a large number of different simultaneous ways of processing input data.

Another very important point in conjunction with systems of this type is the adaptivity involved. What is envisioned is the control of the system adaptively – the sensing of the environment and the illumination of different antenna elements or the transmission of different waveforms adapted to the sensed environment at any given instant. In the next decade or so, I expect this to be a big technological breakthrough in the performance of sensors. To give some numbers which I think are reasonable, a 20-30 DB improvement can be expected in the sensitivity of such radars operating in cluttered environments, difficult environments in which radars are now being demanded to operate. There is also an interaction with certain equipment implementation problems in that sometimes one of the main problems in achieving the theoretically attainable performance is simply the difficulty of maintaining the proper tolerances; for example, the actual element excitations are never exactly as desired.

The system described above is capable of maintaining tolerances because it has a virtual closed-loop control of every degree of freedom in the set. Whether these degrees of freedom represent antenna elements or waveforms or what have you, this closed-loop control is an approach to maintaining tolerances that could not otherwise be achieved.

Now, I shall return in somewhat more detail to the question of satellite communication relays, since this provides another very good example of cases of this type. The multi-access and interactive aspects of satellite communications systems actually exist on two levels. One level might be called the systems level: obviously the servicing of a large number of ground users that have access to an individual satellite communication relay is a classic kind of multi-access problem. In fact, the term multi-access is also used in the satellite communications community, and it specifically means the ability to service ground terminals which, in general, may have a large spectrum of different sizes, radiated powers, and required data rates. It's very well known within the field that it is one of the main problems, if not the main problem, in the implementation of satellite communication relays when the relay is intended to service a user population of this kind. This is especially true, for example, in military tactical communications satellites, perhaps in future air traffic control satellites, and so forth.

The techniques which until now have been explored to deal with the multi-access problem in this context can be described as very primitive compared with what we may expect in the future. Many of the technologies which may be applied in the future to

improve the system multi-access may involve the particular element multi-access level just described – the application of some of these sophisticated techniques to the antenna and associated processing and computing devices carried aboard the satellite. As a matter of fact, I think that communication satellite relays in the future will evolve toward what might be called the great-switchboard-in-the-sky. One key to the application of such a concept is that coverage requirements do not really limit the size of the satellite antenna nor its gain. Up to the present time, it usually is thought that, for example, if a communication system may exist anywhere on the surface of the earth, the satellite beam has to cover the whole earth to cover all of the intended users. This is by no means necessary and would represent an unnecessary limitation on the size of the satellite antennas and their gain. In fact, it recently has become recognized that people are beginning to develop satellite communications relays which have very much narrower and higher gain beams than those which will cover the whole earth. Coverage problems can be solved either by implementing multiple simultaneous beams or by implementing beam steering and switching algorithms; these latter techniques happen to fit in quite well with certain other techniques which have been proposed for the multi-access problem for different reasons, specifically time-division multiple access.

The actual implementation of these techniques would have a variety of effects on the effectiveness of a satellite communication relay. The simplest is that it allows for bigger satellite antennas and hence a system with either larger capacity or ability to service smaller ground users. However, it also provides an entire additional dimension, or two (angular) dimensions to the multi-access problem. Hitherto, systems of this type have regarded the attack on the multi-access problem to consist mainly of modulation techniques; that is, either frequency division or time division or orthogonal codes or something of this sort. The thing contemplated here is to add the spatial directivity dimension to multi-access and this can, in what I regard as practical implementation, easily add another multiplicity of a hundred or a thousand additional channels.

Another effect is that the antenna techniques in which each individual element is regarded as a separate user of a multi-access computer system enables the use of certain techniques such as side-lobe signal cancellation which, in the context just mentioned, involves improved channel isolation for the spatial directivity channels. In the military context, this would translate directly into improved tolerance to jamming, for example, reduction of vulnerability to jamming.

So these are some of the kinds of improvements in both surveillance systems and communications systems which I would expect to become possible by application of this concept of an individual sensor consisting of a computer-controlled collection of many different users. If one looks at each degree of freedom

of such a system as a user, he can easily get into thousands or tens of thousands of users. The saving grace, of course, is that the types of things which users are expected to do may be quite a bit more restricted than in some of the other multi-access systems which have been presented.

There are some problems or challenges in the actual implementation of such systems. If they are to be implemented for sensors with very large numbers of elements – in the thousands – very great requirements will be imposed on computer size and speed. Again, the bandwidth of these systems as well as the numbers of degrees of freedom must be very large. In addition to that, very great challenges are going to be imposed in the development of what might be called the decision and control algorithms involved. Algorithms here refer to the basic rules by which the decisions are reached as to what the different users will be doing at the next moment. There is a very strong relation between the problems of computer speed and size and capacity and the problem of developing the proper decision and control algorithms. There are cases in which the limitation on computer capacity imposes the basic problem on algorithm development. In other words, the development of the algorithms is primarily concerned with the problems raised by computer capacity which is limited in relation to the thing that you are trying to achieve. There are other cases in which actually there's very ample computer capacity. As the jargon goes, the computer resources are ample, but some other kind of resource is limited, such as radiated power; in that situation, the algorithm problem is the development of the proper methods for using the ample computer capacity in such a way as to optimize the use of some other resource which is limited.

In conclusion, the type of problems described here should be regarded as actual typical examples of multi-access interactive computing systems and I predict that the next few decades will bring about some of the most important improvements in both sensor and communications systems, based on applications of adaptive, interactive, multi-access computers in the contexts described.

7. Data Processing Analysis Requirements

Lt. Commander Thomas Knepell

Alameda Naval Air Station

Alameda, California

The state of modern interactive data processing often reminds me of one industrial engineer who defined the systems approach in the following four easy steps: (1) cut to shape; (2) hammer to fit; (3) file to smooth, and (4) draw blueprint.

As a representative user of both business and command-and-control data processing, there is a temptation to challenge the industry for hardware and software improvements. However, the main constraint to achieving the potentials of data processing is that the system capabilities which exist today are not being used.

I do acknowledge the need for improvements in hardware, software, and standards, but for the most part that need exists because the economics of data processing must be improved and not because design capabilities are lacking. For example, today's data communication systems are effective, but it is not yet cost-effective to establish communication networks for all the data which may be appropriate to process. Good time-sharing systems have finally arrived, but effective usage is limited by high cost. Interactive programming can greatly improve program development and maintenance, for those who can afford the price. When will it become economically feasible to have 20 billion characters of online storage for just the basic data needed for an integrated management system? In software, there are now data base management systems, capable operating systems, problem-oriented languages, and more and more software developments, but the operating overhead costs dearly.

This is not to deny that standards and technological advances are needed in hardware and software, but the emphasis and thrust of such developments should be to reduce the cost of today's capabilities, at least until we learn how to use the capabilities which exist now. Others have used the example of Detroit to describe

new computer systems. The analogy is the car with 300 horsepower capable of doing 120 miles per hour, although the roads and the laws preclude driving that fast. High-powered systems are used to keep track of ships at sea, planes on decks, or spare parts in supply bins, just as was done with older card-walloping systems, and sometimes with the same programs. Now, with interactive systems, the ability exists to make the same mistakes faster. The real challenge in data processing is to provide a management or a command and control system, rather than an information system, and this challenge reveals the deficiencies of systems analysis.

As a symptom of the state of the art, I am constantly disappointed that our people cannot evaluate system capacity for multiprogramming computers prior to installation. Simulation packages are available and are being used, but try asking analysts to compare alternative systems with significant differences, where design tradeoffs result in radically different optimization. One finds that simulation criteria have bias toward some specific system design. If the study of multiprogramming system capacity is an elusive science for the field-level analyst, consider the task of analyzing multiprocessors, or arrays of processors, or different levels of memory hierarchy. Here we are with powerful system capabilities, but we don't know how much we need. And after the system is installed, we don't make use of the capabilities we bought.

If we may leave the subject of systems design, let us look at the plight of the systems analysts during the design of computer applications which, after all, is the work for which the system was bought. At one time, the stumbling block for a successful system was the implementation stage-programming, integration, and testing. Those problems still exist, but tools of the trade have been improved. Compilers are better, programming standards can ensure program modularity, data base management systems can handle one of the most common causes of system failure, and automated flowcharts and decision-table coding are available. Computer assistance is appropriate for the stages before implementation, those stages of analysis and design.

In today's environment, a good system application exists if the customer can be provided with accurate and timely status of things. If system designers are clever, managers can even be given status on a selective basis for "management by exception." Today, a successful system application exists if operations can be scheduled on a day-to-day basis, like reconnaissance missions or preventive maintenance. And a very successful, but rare, application exists if the system can be used to feed status criteria into the scheduling system. These are all worthwhile accomplishments. But it is within our technical capabilities and, in fact, we have been promising those who invest in our systems that we will do more. Besides scheduling and updating status, we can and

should provide greater assistance for the planning and evaluation function of management and command. The difficulties, however, are the great number of variables and criteria inherent in planning and evaluation. The solution has been to simplify the problem with assumptions and functions which are sophisticated rules of thumb. In this solution, we do no more than imitate the manager's mental short-cuts without the benefit of his beautiful sense of intuition.

Promising a total, integrated system implies that planning and evaluation applications will be based upon intelligence from the collected data of all variables. The constraint for achieving this promise is the systems analyst who is trying to assimilate, integrate, and use this data within the limits of the machine. This analyst needs better tools than he has today.

The final deficiency of systems analysis which I want to discuss is the lack of criteria for evaluating applications. We do not have the tools to determine the worth of the information the system may provide. Unless funds become unlimited, we will always have to choose among alternative applications for systems. By what criteria of worth will we make that choice? Even more basic, what is the worth of information which determines the choice of investment for a computer system rather than for another ship or aircraft? Determining the value of computer applications should be one of the basic roles of the systems analyst. But we have not gone beyond that stage of relating the value of our systems to the number of clerks that can be replaced or the response time in milliseconds.

If these are the problems of systems analysis for data processing, what are the requirements to alleviate the deficiencies? Certainly, mechanical assistance is an attainable goal for relating user requirements to system design. This assistance should run the gamut from analyzing data elements through system network analysis of timing and data base structures to specifying hardware, and application programs. We also need to improve the body of knowledge of management systems. To start with, basic laws of decision theory, management processes, and their utility must be formulated and applied. Then the common characteristics of system applications must be described so that they can be worked with theoretically rather than instruction by instruction or bit manipulation. Finally, the functions of data processing systems must be related to the basic laws describing computer applications. If we understood applications as well as we do automata theory and information science, then computer systems analysis would be a science rather than an art.

To summarize, the biggest challenge for data processing in general is to draw the blueprint first. Today, the systems analyst cannot specify the system we need, he cannot comprehend all the complexities of the applications for which the system is needed, and he cannot even determine if it is worth the attempt to implement

those applications. The cause for these deficiencies is that we just do not understand the nature of the beast called management information or command and control systems. The potential for using today's technology is mind boggling, but, so far, the only mind to be boggled is the system analyst's.

8. Requirements for an Interactive Modeling and Simulation System

Philip J. Kiviat*

Department of the Air Force

This chapter will deal with requirements at a very general level. Its basic premise is that within our present context, interest, per se, is not in being a programmer, but in being a modeler. There is a difference. Those who have done complex modeling jobs, like modeling industrial or computer systems, realize that programming is not the problem – modeling is. We want to build a system that will help us to be better modelers, partly because we realize that modeling is the crux of the problem, but also because once we decide to use a terminal as our basic vehicle, we know good programming can't be done at a terminal. With the pressure of a terminal I can write five lines of code that are probably pretty good, but I doubt that I can write 100 lines of pretty good code. The pressure is not just economic. It is also social pressure, because unless it is my personal terminal, there is somebody breathing down my neck who wants to get on it.

Once I say I am not interested in programming but in model building, criteria that people generally use to judge programming tools are inappropriate, at least as concerns efficiency. I don't care if a compiler is fast, I don't care how much core it takes – I am interested in something else. I am interested in what Douglas Engelbart** has called "augmented problem solving." I want to somehow help a man articulate his ideas; my basic output is going to be an understanding of the structure and behavior of a system, not its operating statistics. I do want a model that I can run, that can tell me whether rule A is better than rule B. But I am interested in understanding, not data.

Now, that implies modeling a system in its very early design stages, not after it exists. Because after it exists, there is a

---

*Now Technical Director, Federal Computer Performance Evaluation and Simulation Center, Washington, D.C.

**Stanford Research Institute, Augmentation Research Center

- MODELING RATHER THAN PROGRAMMING ORIENTED

  CANNOT DO "PROGRAMMING" AT TERMINAL
  EFFICIENCY CRITERIA NOT APPROPRIATE

- BASIC OUTPUT IS UNDERSTANDING OF STRUCTURE AND BEHAVIOR OF SYSTEM, NOT OPERATING STATISTICS

Fig. 8-1. Basic Premises

different kind of simulation job to do. We are then interested in prediction and in estimation. Here I am interested in finding out whether basic ideas of how a system works are really any good. Professor Corbato implied that he couldn't simulate MULTICS at the very beginning because he knew nothing about it. Well, I contend that if he knew nothing about it, he couldn't design it. But he did; ergo, he knew something about it.

Everyone has at least one idea about how a system of interest works. Some ideas are right, some are wrong. We can generally formulate a model. It may be a very rough model. The data portion of the model may be represented by a few curves that have inflections at different points, that are skewed in different ways. There can be different ideas about priorities in organizations. If we test ideas early, we can generally save money that is orders of magnitude more important than money saved by fine tuning later on. That is why an interactive modeling and simulation system is a good idea; it allows experimentation precisely when there is no time to study a system. The reader should think about most of the projects he has been involved with. If he is in the Air Force, Army, or Navy, he should think about the last large system that was implemented and how much time was really spent designing it, thinking about alternative structures. In one very large Air Force effort, almost the first thing that was done was low-level coding; design was the product of coding and integration. Design is the generation, evaluation, and selection of alternatives, not detailed specifications.

Now, since we are considering requirements, what is needed to do such design? Some kind of translator or interpreter or compiler is required. It can be a meta-assembler, a compiler, or an interpretive system – some way of writing programs. A second requirement is a run-time library that does a lot of work for the user, data generation, for example. These elements are found in every programming language today. A third element is not – and that is a comprehensive run-time monitor that helps in interacting with a system, helps in debugging it, integrating it, and validating it. We can almost say that the benefit of an interactive modeling

- TRANSLATOR – INTERPRETER – COMPILER
- RUN–TIME LIBRARY
- RUN–TIME MONITOR AND INTERPRETER

Fig. 8-2. System Components

system is proportional to the effort spent on the run-time monitor. There is only one effort that I know that is taking a practical look at an interactive modeling system, and that has just begun. It happens to be something within the Sloan School of Management at MIT, the SIMPL interactive modeling system.

Modular programming is one requirement for the system we are designing. For years, there has been talk about modular programming. First, there were subroutines and then integrated data bases. But we still don't have modularity. Still, the word modular, as Professor Dykstra points out, has meaning when modules are constructed in certain very clear, very precise ways. They have to be hierarchical; they have to be completely separable, because one thing we can't do when we sit down at a terminal is to remember everything that is in a program – which may have been written by somebody else. We want a modeling concept that is called "process-oriented." Devised by the Simula group in Norway, it has been adopted by several other programming systems and I have extended it in a design for a new simulation language which I call "Simscript II Extended." It is a broader way of looking at real-time processes than is evidenced in

- MODULES HIERARCHICAL AND SEPARABLE
  - SIMULA PROCESS
  - SIMSCRIPT II EXTENDED PROCESS
- GLOBAL DATA BASE ORIENTED
  - LATITUDE IN DATA STRUCTURES
- LIBRARY CATALOGABLE AND CALLABLE
- COMPOSED OF EXTREMELY HIGH–LEVEL LOGICAL STATEMENTS (CONDITIONAL) AND TIME–DEPENDENT COMMANDS
  - FREEDOM FROM LABELS AND UNCONDITIONAL TRANSFERS

Fig. 8-3. Model Structure

programming systems such as regular Simscript or FORTRAN. It is a better way of modularizing programming; system descriptions can be stored in parts so that a change in one part does not affect anything else. That is, I think, a prime requirement of an interactive modeling system.

Another requirement for this system is that it have a global data base with great latitude in data structure. A major problem with modeling is that most people have to fit their system model to the ideas of FORTRAN, where everything has nice clean edges. It's either a square or a cube. But the world doesn't have clean edges. The world is hierarchical; it's a tree structure, a ring, a who-knows-what. We want to be able to take an idea of a model and very quickly try it out, not spend half our time figuring out how to program it. So we have to have great flexibility and the trend is certainly in that direction. It has to be a global data base because that's how the world operates – many modules all simultaneously accessing the same data. If we really want to talk about pluggable kinds of programs where we can test out ideas very easily, we have to have a library where we can catalogue modules and put them together in a very simple way. Much more simply than most operating systems allow today.

The language itself has to be extremely high-level – I don't consider FORTRAN, for instance, to be a high-level modeling language. It has to be high-level in its logical power – both conditional and unconditional statements – and in its operating commands. We must be able to say we want a process to take place whenever or wherever "the following occurs" or on the sunrise of the third Tuesday after the second rising of the full moon. We should be able to say it that way, if that's the way it's expressed in the real system.

Model verification is another well-discussed topic. It is called program debugging, it is called validation. It is called many things as there are many aspects to it. Generally, we define model verification as proof that what we have done agrees with what we thought we had done and model validation as proof that a model is, in fact, a representation of some reality. We have, first, internal consistency and, second, external consistency. A system can do a lot to assist in these tasks.

First, if the models are right the first time (and the system helps insure this), we don't have too much of a problem. Starting off on the right foot disposes of many problems. So, as much as I would like to get away from programming languages, I can't because I don't know how to design a universal modeling language. Therefore I have to have something that will protect me from my own programming mistakes. I insist that this language have full storage and logic protection to whatever specification I choose. There must be nothing that I can do in a program that is illegal and get away with it; simple things like a subscript out of bounds, data out of range, or an illogical condition happening somewhere must

be prohibited. We must both design languages that minimize sources of error and produce systems that detect those errors we do make.

We must have a powerful interrupt capability that allows us to interact with the model. Let the model specification say that every time it happens that a certain condition exists the program stops and control is passed to the terminal. At that point, I may want to take heuristic actions. I may want to abort an experiment. I may want to take actions at any time under any conditions.

Modeling is a human task, not a machine task. We are simply asking our system to help us interact with a running model on a conversational basis. The best way I can think of to do this is to tell the programming system, "Whenever this happens, tell me about it," and I will decide what to do next.

We must have conditional as well as unconditional ways of tracing because if we want to understand a system we have to have a way to trace data flows on both change and time dependent bases. We can trace on an assignment, or on a change of value that is of interest, or on the occurrence of an event – "Notify me whenever

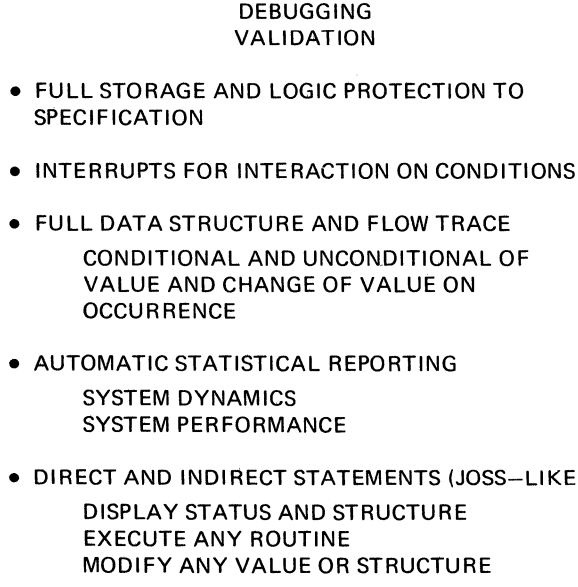


Fig. 8-4. Model Verification

a plane crashes," would be a statement in a model of some military process. "Notify me whenever you run out of stock" and I will see if my logistics algorithm is correct in a specified monitor command in a logistics model.

We don't want to tell the system what statistics we want to look at. It should automatically accumulate, keep track of, and report when we ask all the statistics (like means and variances and histograms) that analysts need. Let the programming system figure out what and how for itself. It's a simple enough job that if we can figure it out, the system can figure it out too. If it doesn't have that capability, it must be programmed.

Last, having been at RAND, exposed to JOSS and deeply impressed, I think that both direct and indirect statements are important. Anything that can be said in a program should be able to be said directly from a console and occur. One can describe the structure of a model, show the status of something or other, execute any statement that will change the model in any way, modify anything, add something to a list, take something out of a matrix, add a row, and change an element. This direct and indirect mode of operation can be extremely powerful in dealing with computational problems and is even more so in dealing in a modeling context. In modeling, we are constantly changing our minds. We begin defining what a model looks like and as we get more and more into the model we realize we know less and less. We are always changing our models. We would like to do it cooperatively and interactively. If we have to reprogram (go back to the beginning and start over again), we might as well be batch processing. We want to change things in real-time; that is, in simulated real-time, to see how the model behaves. Remember, I am not particularly interested in collecting operating statistics so I don't want to go back to the very beginning and start again where I left off. I want to change the model then and there and continue. Let's assume a queue has been building up and I find I have a logical error that caused it. I would like to change the program and let it continue running. If the queue starts to decrease, I have learned something. What its value happens to be at the end of the run is irrelevant; I know that my logic is now correct.

Benefits of interactive modeling are real and important. I claim that the elapsed time of model construction by doing interactive modeling as opposed to conventional batch methods will be in the order of 30 to 1. Only a need for credibility keeps me from claiming 100 to 1, for I believe that with a well designed system we can collapse the time it takes to model systems and study them by perhaps as great a ratio as 100 to 1. I personally encountered a ratio of 30 to 1 when dealing with a system put together using Simscript II on a timesharing system. We had instantaneous turnaround, and could interact with running models, but we couldn't change logic without recompilation. Yet we were doing in a day

what had formerly taken about a month. Thus I feel that with a truly interactive system, this ratio will be even larger. That is important because most of our dollars pay for people, not for computer time. If we cut down on the man hours, we are ahead. A few more dollars in computer time costs far less than the money that a man spends just jingling his coins in his pocket waiting for his job to come back from the computer center. Opportunity costs for good analysts are also large.

Models are of better quality when done in an interactive way because the analyst has continuity of thought and a heightened interest in what he is doing. He can actually see things happening. We found (not unexpectedly) that people were more excited about what they were doing when they didn't have to stop, wait, and start up again. They did more tasks, they did things better, and they experimented more. They had greater insight into a model – its structure and its dynamics, because they were able to play with it. Being able to incrementally construct a model gives a better feeling for what happens when a new part is added. When a model is put together in one piece in the beginning there is little insight into marginal effects. Being able to experiment continuously increases insight. That is what we want.

Better facilities also mean fewer debugging and verification problems. We are now in the loop. Every possible error

- REDUCTION IN ELAPSED TIME OF MODEL CONSTRUCTION ON THE ORDER OF 320:1
- BETTER "QUALITY" MODELS
  - CONTINUITY OF THOUGHT
  - HEIGHTENED INTEREST
  - ABILITY TO EXPERIMENT
- MORE INSIGHT INTO MODEL STRUCTURE AND DYNAMICS
  - INCREMENTAL CONSTRUCTION
  - CONTINUAL EXPERIMENTATION
- FEWER DEBUGGING AND VERIFICATION PROBLEMS
  - BETTER FACILITIES
  - HUMAN INTERVENTION
  - IMMEDIATE RESPONSE TO NEW TESTS AND NEEDS
- REDUCED COST OF EXPERIMENTATION AND ANALYSIS
  - HUMAN INTERVENTION
  - CONTINUING STATISTICAL ANALYSIS

Fig. 8-5. Benefits of Interactive Modeling

condition needn't be programmed. Once in the loop, problems can be found a lot faster and a lot easier. And we can respond immediately to problem situations. We don't have to worry about submitting a number of jobs to beat the system. One benefit doing this will be overall reduced costs of experimentation and analysis. Although the programs we construct will run slower and perhaps take much more core than programs run in a batch system, getting in the loop and being able to do adaptive experimental design will reduce overall experimentation costs and increase the scope of experimentation.

What does it cost to do interactive modeling? It is clear that we have to pay some overhead for timesharing. We have to give the man a terminal. We have to pay connect time, and we have to pay communications costs. For a single terminal we'll pay no less than $200 a month, maybe $1000 a month. If we try to use the system for production work, that is, running simulation experiments to find out exactly how a system will perform down to the last whisker, we have a low-performance system – it isn't designed for that. That's a cost. It will probably cost quite a bit to write an interpreter, monitor, and run time library. What is quite a bit? It could be done in four to six man years with two good people. (I am from the school that says three's a crowd.) If we start using a good existing simulation language as a base, it can be done in less time. Perhaps two to four man years. We also have the problem that programs we produce will not be transferable to other computers and other systems. In some environments this is very important, so I express it as a cost.

It has not been demonstrated quantitatively that interactive modeling is the solution to anybody's problem. Those of us who have experimented in the area feel strongly that we see real benefits. It's hard to construct experiments to demonstrate this. I am 100 percent sympathetic with pleas that we have more measurements to support our contentions but the experiments you have

- TIME-SHARING OVERHEAD
  - TERMINAL
  - CONNECT-TIME
  - COMMUNICATIONS
- LOW PERFORMANCE PRODUCTION SYSTEM
- HIGH COST INTERPRETER – MONITOR – LIBRARY
- NON-TRANSFERABLE PROGRAMS

Fig. 8-6. Interactive Modeling Costs

to perform to find out whether interactive modeling is better than noninteractive modeling are just horrendous. I don't know of anyone who is attacking this problem today. However, I do feel certain that a system such as I have described has great benefits and I am quite as sure that it will be the way we will do our modeling in the future.

PART II. RESEARCH LABORATORY REPORTS

The theme of the three papers in this section is "An Overview of Current and Planned Research and Development Activities plus Some Historical Perspective." This might be paraphrased as "Where is the state of the art?" I have one important point about "state of the art" and this is that the reader be sure he knows which state of the art is being addressed. I have identified at least four. There is the production state of the art, which means it's a catalogue system; it is something that can be bought and essentially the user can install himself. Next, there is the development state of the art, which says that we have all the components and now all we need is some system engineering and integration – somewhat high priced, of course – and we have a system. Then, there is the applied research state of the art, which in general means it works in the laboratory but will it work out in the field? And finally, there is the research state of the art, which says we think we know in principle how to do it. So I caution the reader to be careful about which state of the art is being addressed.

There are several types of research organizations performing research in EDP applications technology; they include computer manufacturers, university computing centers, and private development and systems organizations specializing in the information sciences area. Information from the first two types of organizations appear routinely in the professional journals; however, much of the work done by private research organizations does not seem to get the distribution needed for wide exploitation. The three organizations reporting in this section are among the largest in the field and represent a cross-section of the field. Hopefully, their description of current and planned activities will start a wider distribution of their products.

System Development Corporation is the oldest of the computer-oriented systems and software organizations. Systems Control, Inc. is a relatively new systems organization specializing in the fields of control and automation. Stanford Research Institute, no longer associated with the university, is a major nonprofit firm specializing in surveys and industry planning. Hopefully, the reports from these three firms will be representative of the total field of application-oriented EDP research and development.

9. R&D at System Development Corporation

Clark Weissman

System Development Corporation
Santa Monica, California

The selection of current R&D activities at SDC is organized in this paper by a taxonomy based on processing type. The activities have been placed into five basic categories: data collection, data transmission, data processing, data storage and retrieval, and data presentation.

DATA COLLECTION

A variety of activities exist at SDC in data collection, particularly in man-machine input/output, including English language data management work – the use of natural English as a query and maintenance language, and the extensions of that work into voice research and hand-printed graphics.

This paper discusses three particular areas in data collection. The first is handwritten signature verification, which is representative of the whole general problem of sensor signature recognition. Here the sensor happens to be a human being, and the problem domain we are attacking is that of human verification; that is, a password security approach.

A second class of data collection problem is that of enhancing blurred images from satellite-borne telescopes. This problem is particularly relevant now, as deep space probes become a large part of the space program.

The third type is the use of "smarts" at the terminal level. This provides the ability to perform many data collection functions away from the central computer.

Signature Verification

The principal objective of SDC's work in signature verification is to develop a method for identifying individual human beings.

Considerable attention has been given to the use of voice prints, body odors, and fingerprints, probably all good methods of identification, but impractical and expensive. SDC's approach is based on the assumption that a human being's signature is unique and can be used inexpensively as a unique identifier. The operational concept we have in mind is that the dynamics of the signature can be extracted and matched against data on a private data base. This approach avoids the problem of a large data base, since the data base is distributed in machine readable form on the cards. SDC is in the process of taking signature samples, using a device that can recognize hundreds of characters. With this device we are also exploring the processing strategies for extracting the unique signature parameters. We are investigating the use of low cost X-Y tablets, such as resistive sheets, and the tradeoffs of logic versus verification. That is, the more processing one does the greater is the degree of reliability of recognition; but, of course, more logic is required. Our intent is to get logic cost down so that such devices will be competitive with the kind of verification techniques now in use.

Correcting Blurred Images

The second problem, that of correcting blurred images, is part of a larger task for NASA at Marshall Space Flight Center. The major problem is posed by the many poor images and photographs being recorded. The objective is to discover better techniques for correcting imperfections introduced in the experimental image by the optical devices, particularly where the imperfections, such as jitter in the platform and defocusing, are not known. Device characteristics are described by three-dimensional point-spread functions. The application of two-dimensional Fourier series in the frequency domain tends to reverse the convolution effects in the spatial domain and simultaneously restores the high-frequency data.

Smart Terminals

The third data collection area is that of collecting data at a terminal. The objective of this work is to shift uneconomical tasks from the central processor to the terminal. We believe that today's technology makes it quite possible to integrate a variety of peripherals and minicomputers into a high-performance and surprisingly low-cost intelligent terminal. The operational concept is that of a remote-job station – with data verification, data compression via stored formats, and local buffering – using a multiprogrammed device to overlap the operational characteristics of many of the peripheral devices. It is a flexible device, since these peripheral devices are very easily interconnected. The terminal can adapt to the conventions of the given installation,

which helps to minimize communication costs. An operational prototype consisting of a minicomputer, CRT, line printer, disk and two tape cassettes has been assembled and is being used for experimentation.

DATA TRANSMISSION

There are a number of activities at SDC in data transmission. A few are presented here. SDC has been involved for many years in programmable controllers and data concentrators. The technology was demonstrated in the 1960s and will be quite cost effective for the 1970s.

Another growth area for the 1970s is that of networks of distributed computer resources. SDC has been involved in this since our early (1960) experiments interconnecting the Q-32 time-sharing system with Stanford Research Institute and, later, Lincoln Labs. That work continues with our participation as a node in the ARPA network, in network protocol development, and in network experiments in hardware and data sharing.

Programmable Controllers

The kinds of tasks being assigned to programmable controllers are: (1) line discipline and interface protocols; (2) terminal interface and control – that is, managing the actual physical hardware devices; (3) data concentration via multiplexed low-speed lines and exception transmission of data from preformatted messages; (4) complex data routing and message or line switching; (5) data validation and error handling; and (6) management reporting of device status, message disposition, line conditions, and general system loading.

SDC is involved in development of the Morgantown People Mover, by which the department of transportation is exploring a futuristic technique for moving people on a trainlike "horizontal elevator." A PDP-11 is being used as a programmable controller and communication device that keeps track of the routing of trains to prevent accidents and to keep status information on train location and position for routing and dispatch purposes. A person walking into the station can punch a couple of buttons to indicate where he wants to go, and a small car quickly arrives to take him there.

The ADEPT Time-Sharing System is another example using a programmable controller to handle all the real-time and low-speed terminal communications. A Honeywell DDP-516 computer is employed as a replacement for the IBM 270X series and also serves the real-time needs of our graphics and ARPA Network interfaces. It does many more jobs than were originally intended, thereby satisfying one objective of the installation.

SDC has instrumented a critical diamond freeway interchange in Los Angeles, one of the main downtown intersections, to study traffic flow and eventually develop techniques for improving the flow of city traffic. A Varian 620 is used for transmission processing and control.

In our Air Force Satellite Control Facility, SDC is involved in an application of Digital Scientific's META4, a small machine with microprogrammable logic. It is employed in a firmware approach to "reusable" software, a topic presented later in more detail.

Finally, SDC has a contract with the Los Angeles sheriff's department to automate its communication system to reduce the significant time delays between an alarm made by a citizen and police action, e.g., the routing of a police car.

These are just a few of SDC's new minicomputer programmable controller applications. Each is a comprehensive R&D program, from a few thousand to more than a million dollars in scope. The applications are intricate, and none would be practical at this time without the use of programmable devices.

ARPA Network

SDC is active in the development and operation of the ARPA Network, which has been in use for the past two years. The latest ARPA Network configuration is shown in Fig. 9-1. The ovals are Host computers; the solid lines indicate which of the Host computers are now on the network; the squares are the interface message processors (IMPs), the ARPA Network subsystem; and the dotted lines are the candidate systems, to be joining the Network shortly. The ARPA Net must be viewed as a hierarchy of communication protocols. When the question is asked, "Is it running?", the answer is a qualified, "Yes, depending on what level in the hierarchy you are asking about." Much of the research activity with the ARPA Net is directed toward elevating the hierarchy.

One process-level experiment in which SDC is participating has as an objective the establishment of a network resource center for tablet graphics, so that every node in the net could have the same tablet-graphics capabilities. The initial experiment is to study the different character recognition algorithms that exist in the ARPA community and to start looking at these from a cost-effectiveness point of view. Some of them are of better quality than others and usually run longer, while others are less expensive in runtime and small enough to possibly fit in a minicomputer. Today, there are no criteria for quality of recognition. This then, is a secondary objective that may lead to building an inventory of recognition algorithms. The technical approach of the experiment is first to develop a tablet graphic protocol, and then to share SDC's tablet graphics software remotely from some of the Host sites. We are currently working with the MIT Dynamic

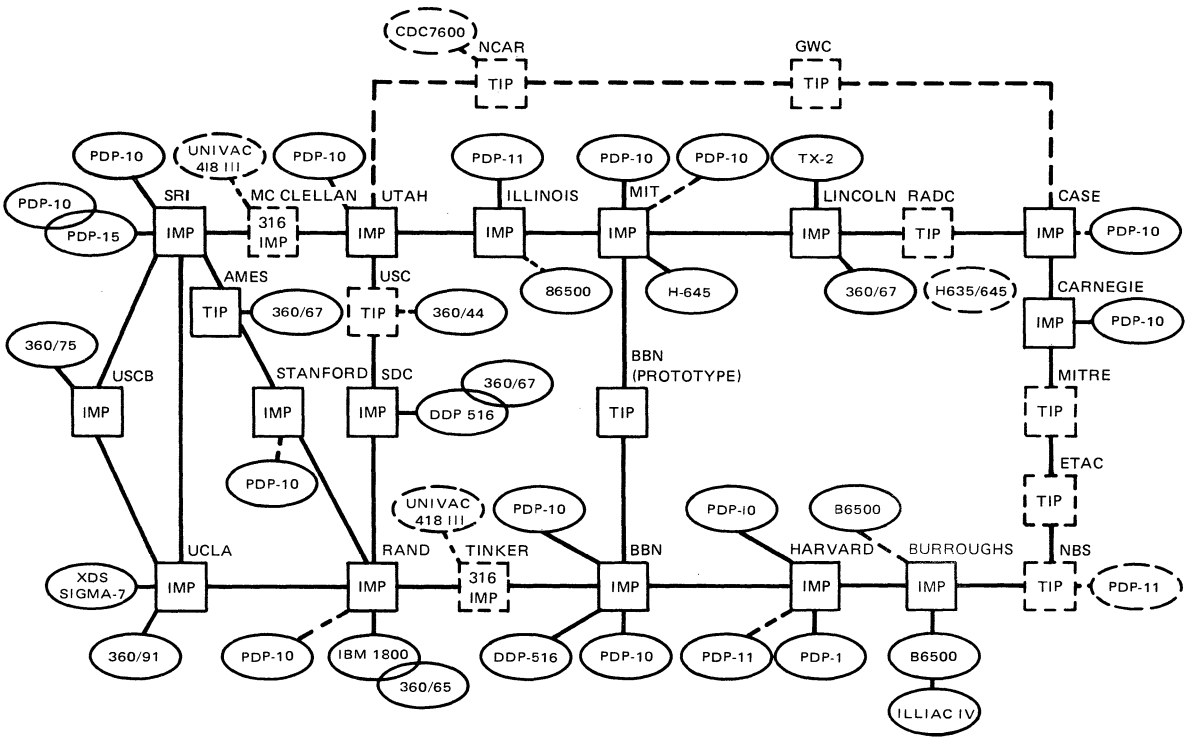


Fig. 9-1. ARPA Net, August 1971

Modeling Computer Group (DMCG). The DMCG RAND-tablet graphic data will be transmitted over the Network to SDC using the data-block protocol. Automatically, the SDC character-recognition routines will process the data and return the identified character, size, and position codes. The present status of the experiment is that the tablet protocol has been specified, implemented, and closed-loop tested, and a universal dictionary has been developed. We are now beginning the live tests with DMCG.

DATA PROCESSING

Data processing will be discussed here in the context of major areas; time-sharing, automated programming, and machine architecture.

In discussing automated programming from artificial-intelligence approaches to programming, SDC is working on automated, not automatic programming; the difference between them depends on whether the human or the machine is in control. In automated programming, the programmer is in control and uses the computer as an aid or assistant. In automatic programming, a computer program does the programming, i.e., the code synthesis – and the programmer's intellect augments the computer to help it out of any crisis situations. The computer is determining how the program should be linked together, based upon internally developed strategies and past learning experiences. SDC is excited about the potential of automatic programming and feels that it is the direction the industry may eventually take. However, its payoff is in the distant future. As a pragmatic organization, SDC is interested in what can be done today and in the near future. Automated programming says that there are many things we know how to do to improve programming that we have not yet done; we have not yet put together the low-cost, quality systems we know how to build. SDC is looking at ways in which that can and has been done, based upon complete software production systems.

In the area of machine architecture, LSI circuitry makes it possible to put computers together for special purposes. SDC is beginning to explore how that should be done in areas that are relevant to problems in improving data processing.

SDC is also active in resource-allocation modeling, particularly in the distribution of resources among nodes in nets. We have built a number of proprietary tools for doing network analytical studies. One of them is called Designet. We are applying these tools in a study for ARPA of communications and computing tradeoffs. The question being addressed is: With lower-cost communications, such as ARPA Net, and lower-cost computing, such as minicomputers, is it better to share large centralized machines or network small machines (perhaps network larger machines, as well)? This gets into the complex cost relationships

among distributed and centralized computing and communications. The ARPA study called CACTOS (for Computation and Communications Trade-Off Studies) is attempting to define and model these relationships.

SDC Time-Sharing

SDC began its pioneering work in time-sharing with the ADEPT Q-32 time-sharing system approximately 10 years ago. ADEPT, of course, is still operating, in a form called the ADEPT Research & Development system, which we are using at SDC for R&D. A version of ADEPT is still performing satisfactorily in the intelligence community, and the Naval Electronics Laboratory Center (NELC) recently received a copy of it. SDC uses TS/DMS, a commercial version of ADEPT, in its service bureau operations.

SDC is currently developing the Interactive Common Operating System (ICOS), which is a merge of TS/DMS and ADEPT into a single system. We intend to run ICOS as a research facility for 10 to 12 hours a day. We are also giving serious thought to tying these machines into a local net at SDC through the ARPA IMP, which can support multiple Hosts. We also have another machine, a Raytheon 704, dedicated to our voice research work; it too will be tied into this local net.

Automated Programming and the "Software Factory"

SDC's main goal in automated programming is to lower programming costs. Two major aspects of our work are formal techniques for compiler implementation, and microprogramming for software transferability.

Although the "software factory" does not exist today, some parts of it do exist at SDC. It is called the "language factory." The industry does not yet have a software factory, primarily because "software" is such a diverse commodity that there is a major difficulty in describing the "factory" clearly enough to convince companies to make the capital investment to build one. The industry needs formal models of its products – be they compilers, operating systems, data management systems, utilities, or whatever. When these models, or formalisms, appear the structure and economic form of the software factory can be defined. Current work on these formalisms is being based on graph structures and directed graphs. I'd like to explain briefly our formalisms for compilers, since we have, we believe, advanced the state of the art in compilers.

SDC formalism, which we believe has advanced the state of the art, is expressed as a model of a compiler as shown in Fig. 9-2.

On the basis of this model, we have developed the Compiler Writing and Implementation Compiler (CWIC), consisting of

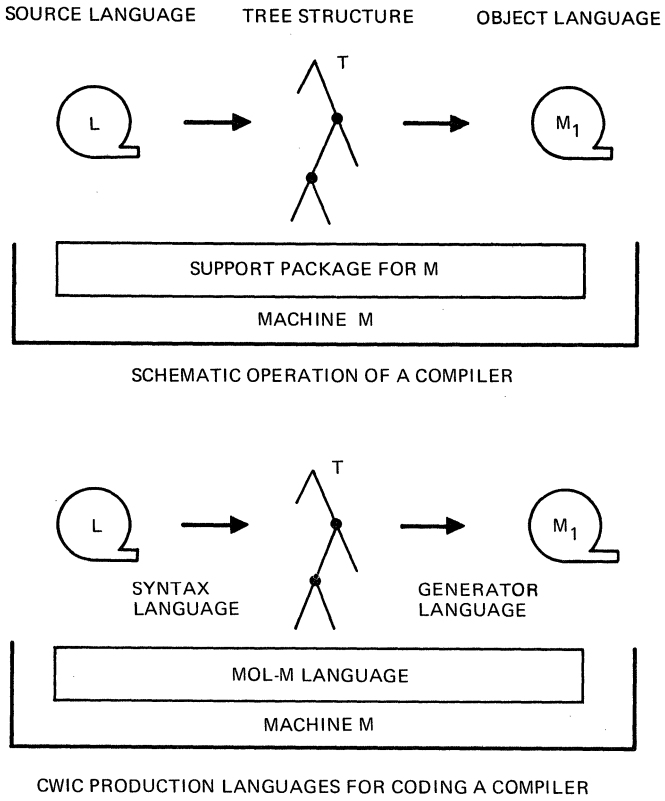


Fig. 9-2. Model of Language Compilation

three languages and their compilers. The SYNTAX language is used by the designer to express the problems of syntax recognition and translation into internal tree-structure form. A powerful graph-manipulation language called GENERATOR expresses the processes of code generation. Finally, a machine-oriented language, called MOL, expresses the machine environment interfaces. MOL is higher-order assembly language with the syntax of a compiler but the semantics of an assembler. MOL is one of the new class of languages called system programming languages, such as PL/360, BLISS, PASCAL, etc. Based on our compiler model and the CWIC family of languages, we can build language software to specifications quite inexpensively. In fact, we often build throwaway languages and compilers for special applications

in a matter of a few days. We are building some substantial languages, as well.

Figure 9-3 shows how we manufacture compilers in our "language factory." First, we assume at the beginning that some supporting routines and a dictionary exist. (If you are running a factory, the last thing you made will be part of the library, and the new support package can be subset from the library.) Next, we write syntax equations – the description of the language we want to compile – and pass them through the SYNTAX compiler. The output is a set of machine-code – in effect, a new syntax dictionary that obeys the specified syntax rules and augments the existing dictionary. Next, we add new MOL routines. (We operate on the IBM 360, hence the MOL-360.) This permits us to extend the basic machine model. Certain compilers need special features which are normally available in the original support package, so we extend the support package. Last, the GENERATOR statements go through a two-pass process. The first pass produces tree-structure representations, and the second pass translates these tree structures into machine-code routines and the dictionary necessary to complete the compiler. Incremental

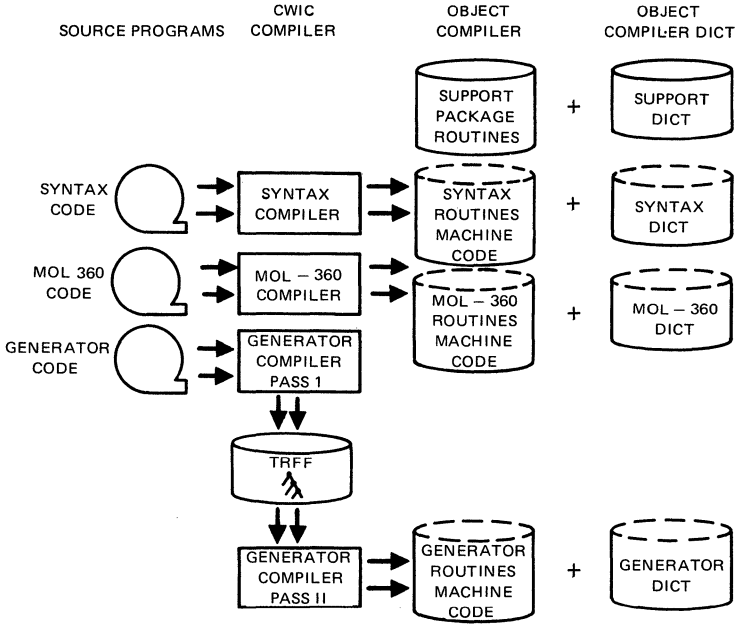


Fig. 9-3. Schematic Operation of the CWIC System

compilation is possible because any part of this process can be repeated at any time; new routines can be added and old routines removed. One man, or at most a few men, can write an entire compiler.

As an example, Fig. 9-4 shows a simple expression-evaluation interpreter that we designed online in about an hour. For the arithmetic expression of X + Y x Z, where X = 5, Y = 12, Z = 5, the tree structure produced by the syntax equation is shown to the right. Below that we have the GENERATOR code to perform the interpretation. It compiles a sequence of program calls to an existing library of routines to perform the arithmetic operations and the interface with the operating system. On this one sheet is a complete interpreter. We are now producing contract software with CWIC and extending our tools to generate code of increasingly high quality.

One of our people has written a book on optimization algorithms that will be published next year. It is a review of the state of the art and includes original work in the area of global optimization. Our system exploits (among others) a concept called "hoisting," whereby common expressions are hoisted out of deeply nested loops. Our techniques work on both the data-flow and control-flow graphs of a program and rewrite the program to improve the code. The GENERATOR language allows immediate implementation of these optimization algorithms.

A version of CWIC system was used to build a compiler for SPL (the space programming language), a cross-compiler system for generating code for airborne and satellite computers. We are finishing an optimized FORTRAN IV compiler using this technique. It happens to be a three-pass structure; passes one and three are conventional, while pass two is an optimization pass. We built it to test our techniques against the IBM FORTRAN compiler, the best optimizing FORTRAN compiler now available. It is now running, and testing is in progress. The total effort involved about one man-year to build the compiler.

The software factory may not be here in all software fields, but it is here in language-oriented software. We are now beginning to understand the formalisms in operating systems and data management systems; hopefully, we can make similar advances in those areas.

Software Transfer by Microprogramming

Microprogramming is another programming area in which we conducted an interesting study. SDC operates a satellite control facility for SAMSO that involves a very large telemetry net and many computers. Some of the equipment is almost 10 years old. Something like a million instructions in the system are in daily use. The intent of the study was to update the net. The development objective was to take one part of the net, a dozen or

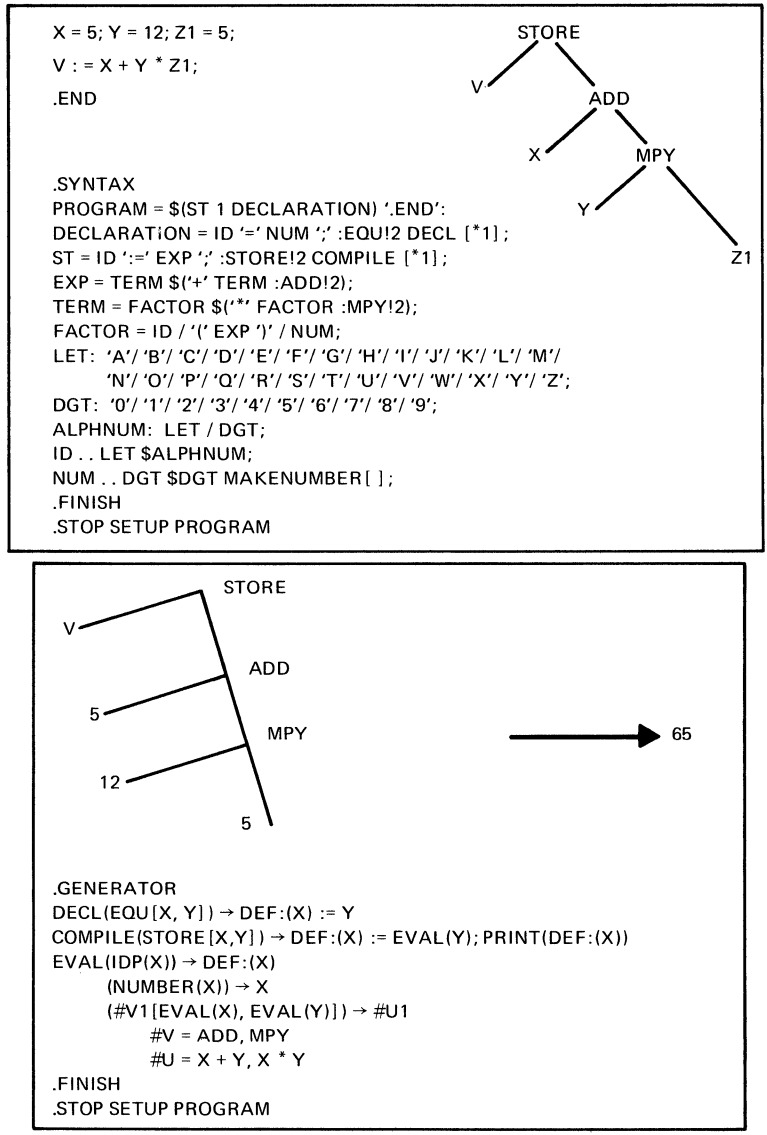


Fig. 9-4. Simple Interpreter Complete Formal Specification and Code

so CDC 160A computers, and improve the system's performance by replacing them without horrendous reprogramming costs. The approach was microprogrammed emulation of the CDC 160A. Constraints dictated off-the-shelf interfacings, no major engineering, emulating the system as is, no great leaps forward. SDC handled the management and design; the actual production of the firmware was subcontracted to the machine manufacturer according to our specifications.

We selected a META4 from Digital Scientific. The equipment has been delivered and is currently being tested. We have enough information to report that there were no major problems except in some very minor, easily repaired, repertoire errors, and some difficulties in timing, since the META4 is considerably faster than the 160A. These were all overcome. In order to determine whether the emulation was correct and more efficient, we traced a few instructions. The data show improvement by a factor of four in instruction execution. The 160A did not have a multiply instruction; one was added in firmware, and multiplication performance immediately improved by a factor of about 46 over the previous software multiply.

Machine Architecture: Associative Processing

SDC is currently involved in several associative processing (AP) projects. Associative processors have a variety of unique and attractive characteristics. Because each memory cell can be viewed as a single CPU, they offer the power of parallel computation, which lends itself very nicely to real-time problems. Because each cell is also an associative memory, storage can be compressed, since common items in a complex tree structure need not be stored redundantly. Fast associative searching results in lower memory and search-time overhead. There is a higher failure tolerance with these systems, since they are not dependent on storage location; memory can have holes in it due to failure without degrading the system. Report by exception is one of the associative operations on these machines. Since it is an associative memory, one can set threshold tests on various limits and have the success cases announce themselves. Of course, AP machines can also perform serial computation, so they can do conventional programming and computing. Two application areas we are involved in are tactical data management and ballistic missile defense.

AP FOR REAL-TIME MANAGEMENT

An AP data management study is just getting underway for Rome Air Development Center (RADC) in conjunction with the Electronic Systems Division (ESD) of the Air Force. The application is to Tactical Air Command and Control (TACC) data

management. The study objectives are to classify and quantify improvements in data management using associative processors, to determine the impact AP machine architecture has on conventional concepts, to look for the critical design factors in such a system, and to look at the basic data structures for such a data management system. The study approach involves three factors: (1) use of an actual TACC scenario as a data management system benchmark of the functions we need to perform; (2) use of an SDC-designed "paper" AP machine for performance simulation of machine factors; and (3) use of the predicate calculus and set-theoretic data structures.

SDC has been working for a number of years in system architecture and machine modeling. We have developed, on paper, an SDC version of an AP machine. We will use it, and extensions of it, as models for testing software construction, comparing coding sequences, and looking at speed tests using computer simulation.

SDC has also developed an English data management system, CONVERSE, which maps English into a formal predicate-calculus intermediate language for retrieval from what is essentially an associative data structure of factual information. The system will be used as a model to see how AP satisfies the internal structure needs of CONVERSE. We expect it to replace much of the list-structure overhead of data management systems on conventional machines.

AP FOR BALLISTIC MISSILE DEFENSE: PEPE

SDC recently completed an initial version of PEPE (Parallel Element Processing Ensemble) for ballistic missile defense. The program objectives are to solve the ICBM threat, which overwhelms even the biggest of conventional machines. There are just too many ICBMs that are too fast. The intent is to demonstrate the feasibility of the PEPE concept to solve the problem. The solution involves handling the acquisition, tracking, guidance and control, and battle-management functions of ballistic missile defense. The technical approach is to allocate one CPU (PEPE element) per target track, and to manage all the elements as an ensemble. That ensemble is connected to a host computer, a conventional sequential processor, which sets up the computations for the PEPE. In SAGE, SDC wrapped "correlation boxes" around unknown tracks to identify friend or foe; those correlation tests can now be made in two associative searches. The interesting aspect of the PEPE is that the elements are driven in parallel so the software is invariant with ensemble size. It has been suggested that PEPE could be an approach to gigantic data base management systems, where one CPU is assigned to each key and the data base is searched, not unlike head-per-track disc devices. But that is for the future.

The current status of this project is that a 15-element PEPE has been built and attached to a 360/65. A pre-specified "zero-order" test was successfully run. The demonstration included working software and a collection of production tools including a parallel FORTRAN, a parallel assembly language, and a number of special-purpose languages. Analytical and simulation studies of large problem applications are currently in process. (That is, analysis is already going on for the next generation of PEPE.)

Figure 9-5 shows the serial host connection to PEPE. The host is currently a 360/65, but it could be a 6500 or a 360/195 supercomputer. Hanging onto the channel interface are the two parts of the PEPE: the Correlation Control Unit and the Arithmetic Control Unit. The correlation units perform the associative operations of the correlation boxes; e.g., "all tracks that are between certain altitudes identify yourselves." Conventional processing is performed by the arithmetic units. A PEPE element consists of 8 words of 40 bits (or one 320-bit word) of associative memory; 512 32-bit words of conventional memory, and a collection of arithmetic tag registers for doing the arithmetic operations. In this particular PEPE design, arithmetic activities are done outside of the memory, so there is actually an internal transfer to the registers. Newer designs of PEPE elements may obviate these transfers, since memory logic and computation logic will be core resident in memory.

Figure 9-6 gives a gross cost/performance tradeoff. The number of targets appears on the bottom axis and million instruction executions per second (MIPS) on the vertical axis. Data show that the limit of a 6500 computer for this task is about 14 MIPS. This would yield a 70-target capacity for a single machine. To increase the track capacity to 230 would require approximately 47 MIPS, which would require three 7600s. If you offload all of the tracking problems to a 230-element PEPE, attached to one 7600, at $9K/element, plus the $8 million cost of the 7600, the PEPE solution costs about $10 million as compared to $27 million for three 7600s – quite a cost-effective tradeoff. This cost-improvement factor of 2.7 is a conservative estimate, since the multiprocessor 7600 configuration is nonlinear in cost/performance and may need a fourth CPU to control the other three.

DATA STORAGE AND RETRIEVAL

Over the years, SDC has developed a variety of storage and retrieval systems. A-32 LUCID led to TDMS, which led to CDMS (a commercial version of TDMS), which led to SACCS/DMS, an advanced TDMS version that successfully attacked the real-time-updated problems. We have a number of other DMS products. DS/1 and DS/2 are "mini" versions of TS/DMS. They have the language flavor of TDMS but work on small machines (a Model 30) using serial files. SDC developed another package called Orbit

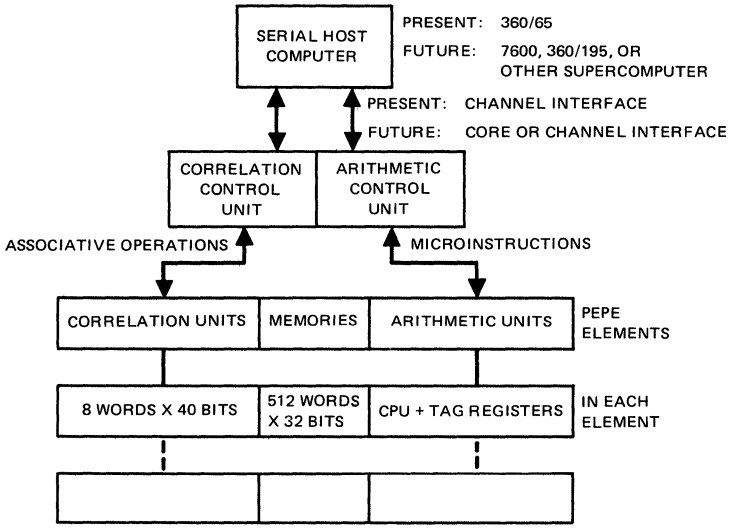


Fig. 9-5. PEPE Hardware

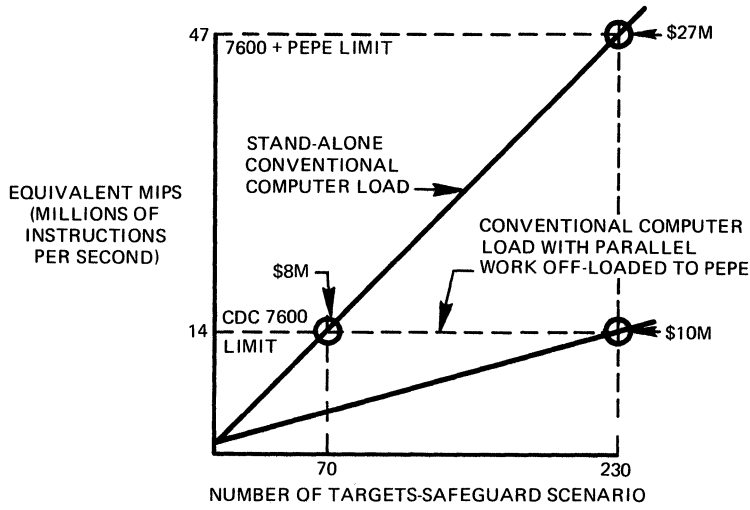


Fig. 9-6. PEPE/Host vs. Host – Only Performance Trade-Off

(Online Retrieval of Bibliographic Text) that is currently providing users of the National Library of Medicine with nationwide time-shared access to medical records. Further developments currently underway include the DS/3, an extension of DS/2 that handles indexed files, and three activities which will be described more fully: MEDLARS II and SEAWATCH, two larger-scale data management systems; and a research study into network data sharing.

MEDLARS II

MEDLARS II is being built for the National Library of Medicine to allow the online input, editing, storage, and retrieval of medical bibliographic citations. These are large files with millions of citations. MEDLARS II will also build reports for users. The objectives are to provide high throughput and large capacity service, thereby yielding a better balance between the people who do the cataloging and the people who do the indexing, and allowing remote interactive file search for a national user community. The activity has just begun, and is building on our ORBIT experience. MEDLARS II will be implemented on the 360/155. It is coded in PL/1 and Basic Assembly Language (BAL) in modular form for future expansion.

SEAWATCH

SEAWATCH is an ocean surveillance intelligence data management system that will support both batch and interactive operations on numerous massive files. Our development approach builds on the TDMS and SACCS/ DMS experience. SEAWATCH is being implemented in JOVIAL on the CDC 6400 under the Scope 3.3 operating system. Structurally, there are seven subsystems and 38 functions that service 84 static, dynamic, historical, and working files. Both serial and inverted tape and disc files are involved. It is a comprehensive system. The specifications and the design are complete. Software for the initial operating capability (IOC) is in production, and IOC testing is scheduled for the spring of 1972.

DATA SHARING IN COMPUTER NETWORKS

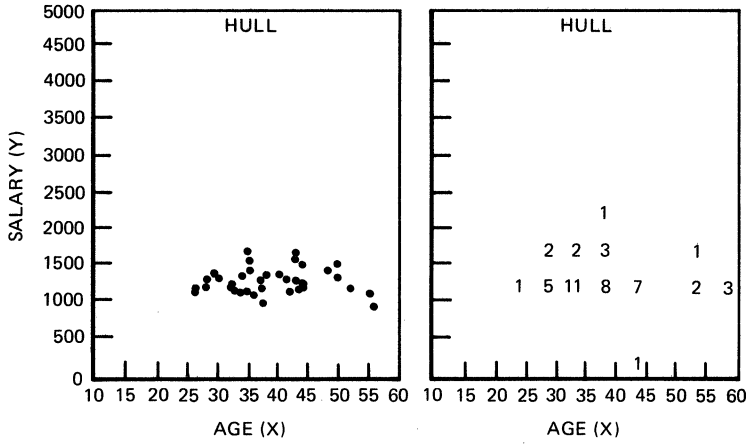
A study now in progress for ARPA concerns the integration of dissimilar data management systems in digital networks – today's systems and those of the future. One approach to this problem is to leapfrog today's problems by designing tomorrow's common system. The overwhelming problem is not data management but the corpus of the data bases that are being built and are already in existence. Conversion is prohibitively expensive today, and the cost will increase as data management systems become more widely used. SDC's assumption, then, is that today's

dissimilar data base systems are going to be around tomorrow, and that we must find ways to share them. The ARPA network and other similar networks allow access to host systems, but their data management systems are essentially unavailable because there are not convenient ways to interrogate them. The user certainly does not want to learn 20 or more different data management languages (DML). SDC's approach is to explore English and the predicate calculus intermediate language (IL) of CONVERSE as a possible common network data-sharing language. We also intend to employ compiler-compiler techniques to build an IL-to-DML translator for each host data management system. We are technically able to build these translators at low cost; we are now exploring whether the predicate calculus is rich enough to express the queries.

DATA PRESENTATION

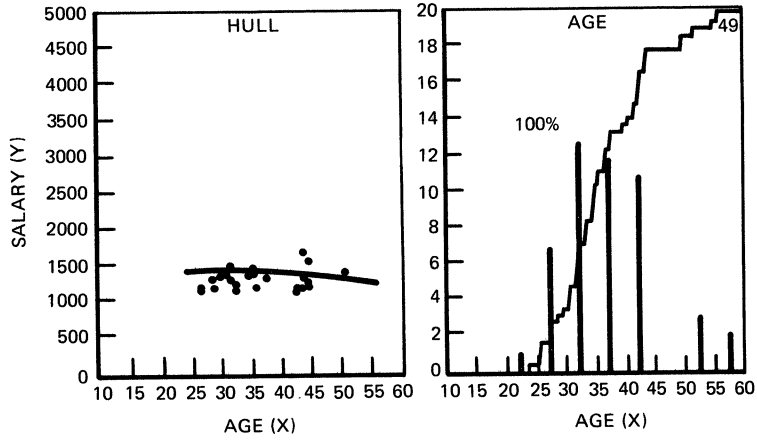
The data presentation topic covered will discuss ARPA-funded work in data management graphics completed last year. The intent was to provide graphical capability to the nonprogrammer user who manipulates data – the form, the format, and the content of the display. Also, the data presentation system was to be a front end to an existing data management system. SDC's approach uses an ARDS storage-tube terminal, a low-cost graphics device. To keep interaction simple, the man-machine dialogue is totally by selection and prompting. Also, the computer automatically scales all axes of all graphs. The data management system independence is achieved by use of an "ancestor file" – a subset file extracted from the larger DMS data base.

This system is complete and operational. Figures 9-7 and 9-8 indicate its graphics capability. The left side of Fig. 9-7 is a scatter plot; the right is an interval plot. Users can specify data to be plotted. Figure 9-8 shows age vs salary, in two other forms: a regression curve and a histogram. These are typical of things done "ad lib" at the console.



PRINT	BUILD	GRID	AXIS	PRINT	BUILD		AXIS
PLOT	NEW-DATA-BASE		READOUT	PLOT	NEW-DATA-BASE	SCATTER	READ-OUT
SAVE	HISTOGRAM	BACK-UP	TITLE	SAVE	HISTOGRAM	BACK-UP	TITLE
RESTORE	REGRESS			RESTORE	REGRESS		

Fig. 9-7. Scatter Plot of Salary Versus Age + Interval Plot of Salary Versus Age



PRINT	BUILD	GRID	AXIS	PRINT	NEW-DATA-BASE	HISTOGRAM	
PLOT	NEW-DATA-BASE	SCATTER	READOUT	PLOT	SAVE	AXIS-SCALE	TITLE
SAVE	HISTOGRAM	BACKUP	TITLE	BUILD	RESTORE		
RESTORE	REGRESS						

Fig. 9-8. 3rd Order Regression Fit of Salary Versus Age + Histogram of Age

10. R&D at The Rand Corporation*

R. H. Anderson

USC Information Sciences Institute

This chapter will discuss some of the ARPA-sponsored research at Rand involving multi-access computing. I hope to present this research program in such a way that the underlying philosophy we have about multi-access computing becomes clear and to show the underlying unity behind several different research programs.

A discussion of technological advances in information science isn't sufficient. There are too many examples within research laboratories of advanced technology which is not being made available to the military. I am therefore including here some thoughts on the transfer of technology from R&D laboratories to operational users within the military and also on the organizational structure necessary within the military to take advantage of and fully utilize the types of technology presented in this text.

MULTI-ACCESS COMPUTING RESEARCH AT RAND

Research at Rand involving multi-access computing is based on the following underlying philosophies:

1. Software packages for online systems, which are attractive and useful to people other than programmers, should be developed.

*Any views expressed in this paper are those of the author. They should not be interpreted as reflecting the views of the Rand Corporation or the official opinion or policy of any of its governmental or private research sponsors.

2. The consoles at which interaction takes place should be personal terminals – usable within a person's office, his best working environment.

3. This personal terminal should be a generalized terminal; from this terminal, the user should have access to all online systems in which he is interested and he should be able to communicate using whatever input and output devices are most appropriate for this purpose.

Specific examples of Rand's research in multi-access computing are presented in the following paragraphs.

The Rand Video Graphic System

The system which best exemplifies our approach to online computing is the Rand Video Graphic System. This consists of up to 32 interactive graphic consoles with each console permitting a full range of interaction from static displays to dynamic interaction with full graphic capability. From this console, a user may communicate with one of several computers and one of several services which might be resident in each computer. Furthermore, the cost of each individual terminal is sufficiently low that a user may treat his terminal as personal property. The terminal's design also permits a wide range of input devices ranging from keyboards to light pens and tablets. Figure 10-1 shows a console of the Video Graphic System. It is based on an 873-line standard television monitor for graphic output. This monitor can display up to 52 lines of text, each line containing 74 characters. This same monitor will also display continuous or dashed lines in three intensity levels. Continuous-tone pictures from a TV camera can be superimposed on a monitor along with computer-generated text and drawings. Since the display is refreshed at a constant 30 frames per second, the display is flicker-free regardless of the amount of information displayed upon it. Figure 10-2 shows the configuration of the Video Graphic System. An IBM 1800 computer is used as a message switching system to allow any terminal to connect dynamically to any computer at Rand. Through the 1800, a terminal may also connect to the ARPA Network of computers, thereby giving the user at one console access not only to the computing facilities at Rand but to those of any computer system on that Network.

PRIM

A second multiprogramming project at Rand is called PRIM (for Programming Research Instrument). The object of this project is to provide a powerful microprogrammed facility which is available to many users within a multiprogrammed system. There are several important unique features about this project.

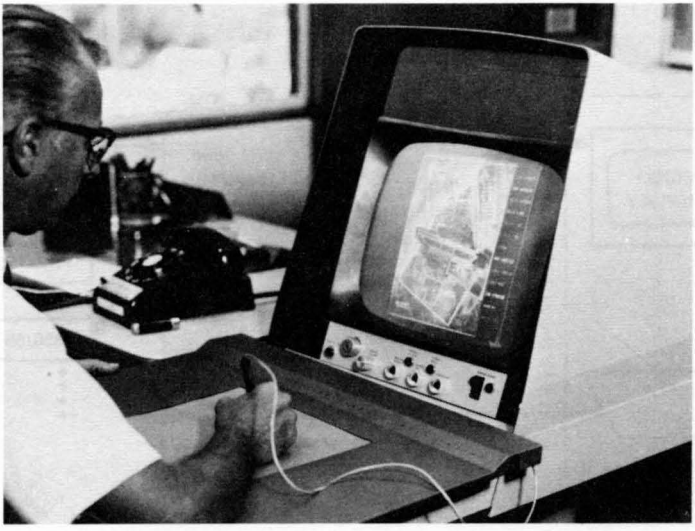


Fig. 10-1. Video Graphic System Console

A microprogrammable CPU with writable control store is connected through a large-capacity, multiported memory to a conventional computer – in this case, a PDP-10. Figure 10-3 shows the configuration of these machines. There are paths of direct communication between the microprogrammable CPU and the monitor CPU. Multiprogramming is accomplished by the PDP-10 acting as a monitor, and it is responsible for scheduling access to the services of the microprogrammed CPU. Task switching is similar to the switching which occurs among users of a conventional timesharing or multitasking system. Switching takes place at "natural" breakpoints (i.e., waiting for terminal input) and at "forced" breakpoints according to an algorithm designed to maintain desirable response characteristics. The monitor engine handles all terminal and network access to the facility and handles primary and secondary storage management and all compilation, program editing and debugging services for the service. The microcode store for the microprogrammed CPU is swappable from the shared core and can be swapped with completely new control code in about 2.5 milliseconds; to swap between two different users who are using common control code in the microprogrammed facility, the swap time is about 0.5 millisecond. The control store for the microprogrammed CPU contains 5000 words: 4000 words of control memory, and 1000 words of registers (status, flip/flops, and relocation constants.

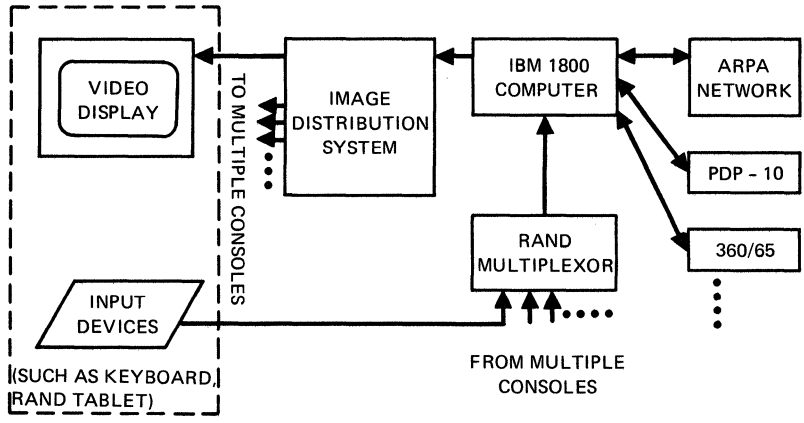


Fig. 10-2. Video Graphic Hardware Configuration

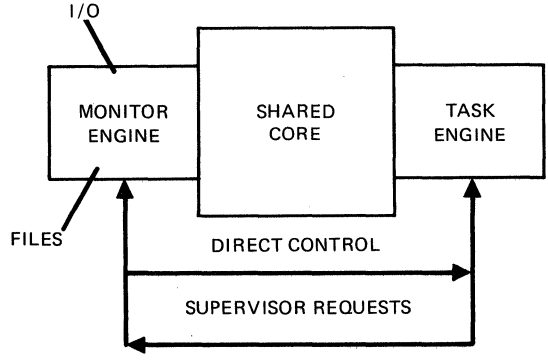


Fig. 10-3. PRIM Configuration

Software

Flexibility can be obtained in hardware through microprogramming, as described above. We are also trying to obtain a maximum degree of flexibility in our software systems. As mentioned earlier, one philosophical underpinning of our research is the desire to put computing power in the hands of nonprogrammers. The traditional reason for the importance of having a programmer at the interactive terminal is the stupidity of conventional computing systems. These systems must be instructed very precisely in a very exact language – and that is the process of programming. At Rand, we are trying to apply some of the techniques of computer science known as "artificial intelligence" to make our interactive systems seem more intelligent to the user. For example, an intelligent terminal might be instructed by example or by analogy rather than by requiring explicit instructions. Two software research projects attacking this problem are CASAP (for Computer Assisted Specification of Algorithmic Processes) and The Adaptive Communicator Project.

CASAP

CASAP attempts to simplify the task of directing a computer by providing the user with a form of specification as close as possible to that used between two intelligent humans. As such, CASAP is concerned with problems of understanding directions in a certain context; submerging details and filling them in; assimilating new concepts; requesting further information in ambiguous or undefined situations; and integrating previous process specifications and adapting them to a new task or environment. Crucial to this project is the problem of representing information within a computer in such a way that relevance of information can be determined and so that general concepts can be used in specific instances which demand some modification and adaptation.

The Adaptive Communicator

The Adaptive Communicator project attempts to construct a flexible man-machine interface, not by detailed programming, but rather by having the user give examples of correct interface behavior and for the interface system to automatically construct from those examples rules of behavior which will govern its interpretation of actions in the future. The basic approach of this project is to store all behavioral information acquired from the user in the form of pattern-replacement rules which can easily be added, deleted, and modified within a set of such rules. This underlying organization allows a flexibility in the interface which is not possible if that interface is coded within FORTRAN or some other traditional programming language. A prototype system based upon these ideas has been constructed in LISP and is currently undergoing tests.

Computer Security

In any discussion of multi-access computing systems for use in military environments, the question of system security soon arises. Invariably, there is sensitive information within the system whose distribution should be limited. Ideally, SECRET information should be able to reside in a system to which uncleared persons have terminal access, with sufficient hardware and software controls to guarantee correct access to all users and information files.

The traditional approach to the security problem has been physical security – having physical control over all terminals and I/O devices for a system in which classified information resides. However, it is becoming increasingly difficult to permit equal clearance of all users accessing a system at the same time, and to insure physical control of all facilities accessing that system.

A solution to this problem involves the deeper question of the possibility of operating system certification. Rand is currently engaged in a major study of system certification, concentrating on three important aspects of the certification process:

1. Assuring that the design of an operating system (and of the hardware features it relies on) is sound,

2. Assuring that the implementation of that design is accurate – that the design has not been compromised either by coding errors or by deliberate attempts to subvert the design,

3. Assuring that the system currently in operation is in fact exactly the system which was designed and implemented – that it has not been modified in any unauthorized way.

These are very difficult problems. We hope that the results from our study of them will allow multi-access computing in many areas where it is now difficult or impossible for reasons of security.

ARPA Computer Network

Another area of multiprogramming research at Rand involves work on interfacing our computing facilities to the ARPA Network of computers. We have constructed the interface through our Video Graphic System in such a way that any of our 25 Video Graphic terminals has complete access to not only the computers at Rand but, in addition, to the full facilities of the ARPA Computer Network. We are currently experimenting with passing graphical information over this network and have been communicating with the University of California at Santa Barbara in an experiment to

use one of their interactive computing systems, the Culler-Fried system, with a graphical interface (including tablet input) on a video terminal at Rand. This experiment has been quite successful and the interaction is as natural as if we were using one of our own computers to do the computation.

Summary

To summarize this brief survey of multiprogramming research efforts at Rand, we are fully committed to the concept of personalized, generalized terminals which have access to a wide variety of computing services, and we are working on software techniques to make these services available to nonprogrammers in such a way that the terminal seems like an intelligent servant which can be instructed in a rational manner. We feel this emphasis is most important in systems for military operational use. By attacking the problem of system certification, we hope to allow multi-access computing to exist in many environments in which it is not now possible.

Up to this point, I have concentrated on new technology being developed by our research program. But technology really isn't the problem. The problem is transferring this technology out of R&D labs and into the hands of operational military users. In addition, once information systems have been disseminated to the military there must be appropriate management structures in the Armed Forces to insure the proper integrated use of this technology.

MANAGEMENT OF INFORMATION SCIENCES WITHIN THE MILITARY

With the increasing emphasis on command and control systems and more sophisticated data collection through various sensors, information systems are having an increasing impact on the way the military performs its role. To help assess how these information systems should be treated within the military organizational structure, I propose to use as a model an area where measures are more clearly defined – namely profit-oriented industry. There, the ultimate measure of how effectively information systems are being used is the profitability of the corporation. First National City Bank in New York has a "computer-smart" group vice president; Eastern Air Lines has a vice president for computer sciences. We believe that by 1980 most major corporations will have a vice president for information sciences, and major decisions will be based upon use of information systems. Note that for a vice presidential post to be filled by computer people, it is necessary that the chief executive officer of these corporations have an understanding of the importance of information science.

Prior to the creation of these high-level posts, computer expertise existed within autonomous centers within the corporate structure. In most cases, costs soared, there was a lack of central control over computing within the entire corporation, and many opportunities for the profitable employment of information systems were lost because no one at a sufficiently high level within the organization had the charter to perform these functions. Over time, responsibility for corporatewide use of information sciences has escalated to top level management.

If industry is right in this trend, the military should pay attention. The benefits of having "computer-smart" persons at a high level are interesting and profound. Within the military organizational structure, a high-level information specialist could provide both global and local guidance to contractors and research labs for computer-related development projects. Also, lower level decisionmakers would have someone with whom to communicate at higher levels without jargon-related barriers.

In cooperation with the Air Force Scientific Advisory Board, members of Rand's Information Sciences Department have proposed an Assistant Chief of Staff for Information Sciences for the Air Force. I am not aware of any final action to date on this proposal, but it might serve as a model of what other services should consider in this regard. We believe such a post is justified within military services based on the amount of money expended on information sciences within the various services and on the importance of that technology to the performance of all missions within the services.

TRANSFERABILITY

In conjunction with the problem of effective management control of information science, there is the problem of transferring technology from research labs into operating agencies. For research ideas to become usable in the field, they need development, production engineering, and production, not unlike the steps a commercial product must go through before it can be manufactured. Most R&D labs are not equipped to carry research through this development chain. The burden should also not be put on operationally oriented users. The military is a prime supporter of research and development. It deserves the first crack at technology which it has underwritten, but it is currently not getting it – mainly due to the lack of an adequate transfer mechanism. Much transfer currently comes from private industry. It picks up some research and development and develops it on its own, but in the process that development becomes industry-oriented rather than military-oriented. The process of transferring research into this field should be controlled to a greater extent by the military. Currently, however, this charter does not seem to explicitly belong to any particular agency. I propose that as a step toward the

solution of this problem the charter of agencies funding research be broadened to allow the funding of development of that research and its transfer into operational use. Much of this transfer process could be done by direct subsidy to industry, but it takes wisdom on the part of this agency to know what developmental work industry will do on its own (and therefore need not be subsidized by the military) and what, out of the various research ideas existing within the lab, are the ideas that should be pursued and are candidates for development and eventual use. Again, having the correct management structure would create a high-level center of expertise of information sciences which could be a source of wisdom in these areas.

CONCLUSION

In summary, we have some interesting research ideas – a Video Graphic System permitting up to 32 users to have sophisticated graphical interaction with a system; we have a microprogrammable machine available to different online users; we have software techniques under development for making computer systems smarter and easier to talk to; and we are tackling the problem of the certification of hardware/software systems for multiuser access.

We know of application areas within the military for these systems and ideas, and we are actively pursuing the transfer of these ideas into operational use, but we neither have the facilities for carrying these ideas through the entire developmental process, nor are we often given the charter to do this. Until the military realize how fundamental to their mission the proper treatment of information systems is, and until they develop an integrated approach toward research, development, and use of these systems, there will remain a wide gap between the facilities we develop at places like Rand and the facilities which are available to the operational users who most need them.

11. R&D at Stanford Research Institute

Marshall Pease

Stanford Research Institute

Menlo Park, California

This chapter will review the work at Stanford Research Institute in information systems, covering first artificial intelligence and then discussing in more detail the research currently in process on Large File Management Systems.

With the program of the Augmentation Research Center, described elsewhere, we have a spectrum of research. The Artificial Intelligence Center is concerned with making a machine solve problems with a high degree of autonomy. At the other end of the spectrum, the Augmentation Research Center is concerned with providing the tools whereby humans, either singly or in groups, can better solve their problems. In between, the Large File Management program has as its general objective to make a creative synthesis of man and machine so that together they can solve problems.

ARTIFICIAL INTELLIGENCE CENTER

Stanford Research Institute has conducted research in artificial intelligence for about ten years. The program was started by Dr. Charles Rosen and is currently under the direction of Dr. Bertram Raphael. The basic goal of the program is to study problem solving and perception and to discover ways in which these capabilities can be automated.

Robot Research

The most visible part of the artificial intelligence program concerns a robot, in the form of a mobile vehicle that carries a television camera. The objective of the work with the robot is, first, to understand what elements are involved in problem solving and perception for a robot; second, to determine how these can be

implemented in the robot's environment; and, third, to determine how to realize the necessary algorithms as a set of effective machine programs. The robot is intended to deal with problems that, in human terms at least, are very simple. Specifically, it is given a task such as that of pushing a particular box into a corner of the room. Its strategy is to investigate the nature of its environment through a television camera and to analyze this picture into a model of its environment. Having completed this analysis, it proceeds to develop a strategy for achieving the goal and then seeks to execute that strategy. In the course of this execution it may discover something it has not recognized before; for example, that another box is in the way. When this happens, it seeks to incorporate this new information into its internal model of the environment and to revise its strategy for accomplishing its goal. It then returns to the execution mode. It is quite successful in performing such tasks in this environment, although admittedly it uses a great deal of computational power and much memory. One of the principal research tasks is to discover how to reduce the computational and memory requirements so as to permit the undertaking of more complicated tasks.

One area of research on the robot has been to improve its perception of the environment. How should it process the complex, ambiguous television picture that it picks up so as to produce a valid model of its environment? One possibility currently under investigation is the addition of other dimensions to the perception process. For example, this might include the use of color or of means for a direct measurement of distance.

Another research area has been concerned with improving the effectiveness of the problem-solving algorithms. One way of doing this is to set up the system so that it learns by experience. After it has, for example, solved the problem of pushing a box into the corner, we would like to have it remember how it accomplished the solution so that in the future it will not have to solve the problem from scratch again. The obvious way is simply to store what it did. However, this so specializes the solution and becomes so highly dependent upon the particular details in which the task is undertaken, that the solution is almost worthless. The approach being pursued instead is to store a generalized form of the solution expressed in terms of various general parameters rather than specific locations. The key information is how the problem was broken down into subgoals. A large part of the strategic analysis undertaken on any problem is the development of a sequence of subgoals. For example, to push the box to the corner, the robot must place itself in position to push the box. The attainment of that position becomes a subgoal. It is possible to analyze a particular strategy in terms that are quite generally parameterized and to develop and store the list of subgoals and results. Somewhat unexpectedly, this approach is not much more difficult to realize than solving the problem in its specific forms as it exists

at the moment. This approach does lead to the possibility of developing a repertoire of generalized strategies. Hence, the robot is enabled to learn by experience.

Speech Understanding

Another area of research that is just getting started in the Artificial Intelligence Center concerns speech understanding. The goal of this new program (funded by ARPA) is to allow a person to talk directly to a computer. This may be considered a problem in artificial intelligence, since, when someone speaks in a fairly free form of English, the selection of the appropriate interpretation from the set of syntactically possible meanings is a very complex problem-solving task of the type often found in artificial intelligence work.

This work relates rather closely to the continuing activity on various Q/A (question/answering) languages for use with inferential file systems. This activity is deeply concerned with the representation and analysis of natural and nearly natural language.

Problem Solving Methodology

Another major area of activity is research on automatic programming and theorem proving. This is very much at the heart of much artifical intelligence work. It is the basic approach used for problem solving in the robot. Given the problem of pushing the box into a corner, the system sets up a "theorem" that the box is in the corner. Finding that this cannot be proved, since it is, in fact, false, the system considers what the state of the system would have to be for the theorem to be provable. It considers the transformations that would lead to such a state. It considers a new theorem that one of these states does exist. If this theorem is provable, then the system has a strategy for achieving the desired goal. If not, it considers again what conditions would lead to the realization of one of these states. It continues to work backward this way until it has finally reached a provable theorem. From this state it can then work forward to develop a specific strategy that will lead to the desired goal if its understanding of the environment is sufficiently complete. The various states that make the successive theorems provable constitute the series of subgoals leading to the desired goal. This may seem like an indirect way of handling the problem, but is an effective procedure – and, most important, a general one – that can, in fact, be implemented on existing computers.

Applications Studies

The final area of activity that I will mention is an effort to develop a program for applying artificial intelligence capability

within industrial environments. There are, of course, highly automated activities within industry handling repetitive jobs. The problem that we see here is that such machines function only in a very tightly controlled environment. It should be possible to use artificial intelligence procedures to expand the capabilities of these machines so that they become goal-seeking within a much more variable environment. One aspect of the problem is the need to add sensing capability that will allow the system to perceive the environment as it exists, together with the capability of analyzing the perceptual data into an effective model of the existent environment.

LARGE FILE MANAGEMENT RESEARCH SYSTEMS

My own area of primary interest, Large File Information Systems, is a program started about ten months ago under ONR sponsorship. It is a long-range task aimed at the problem of providing computer support for management decisions. The goal is to discover principles, and means for their implementation, that will enable a system to provide interactive assistance to the manager in his decision-making role.

The thought that motivates the program is that good managerial decisions require a very subtle integration of large and variant data files. The relevant data bases may be so large and complex, contain so many different types of data, and have inherently such great complexity of interrelations as to pose a considerable challenge to present-day computer power. The questions being asked may be subtle, involving considerable inference on the specific data elements. The design of the system that will handle such a large and complex data base and that will be responsive to the needs of a manager probably represents one of the most difficult challenges facing the system designer at this time.

There are, of course, many systems in existence that are intended to aid the manager. However, I do not think many would disagree that very few of the existing management information systems have lived up to their initial promises or achieved the degree of effectiveness that should be possible. Few, if any, are used by managers in their day-to-day decision making.

Bottom-Up Approach

It is evident that many existing systems have had their origin in the fact that there were certain operations in an organization that could profitably be automated. The first goal of the system, therefore, was to automate, for example, the accounting system, or payroll, or inventory control. Managerial information, in the form of routine reports, is then, typically, built on top of this to the extent that it is implemented at all. This does not deny that

the automation of routine operations is valuable, but such automation has very little to do with managerial processes. The key to all truly managerial action is precisely that it is not routine. The manager, when he is functioning as a manager, is always dealing with the exceptional situation. By definition this is nonroutine. The manager delegates routine to the subordinate and confines his attention to the unexpected and unusual.

Top-Down Approach

So far, I have outlined the evolutionary procedure that is sometimes called "bottom-up" design. The alternative approach has been called the "top-down" approach. In this approach, an analyst studies the organization and tries to determine what information the manager needs. He then seeks to build a system to supply that information on demand. The difficulty with this approach is two-fold. First, it takes a substantial period of time to build the system (probably two years is minimal). This means that when the system finally becomes operational it is responsive to needs seen a substantial period of time before, but which may very well no longer be applicable. Of course, the perception of needs may have also been wrong to begin with! Second, the system depends upon predicting the requirements of the manager. It is characteristic of the managerial situation that the environment in which the manager functions and the problems with which he deals change very drastically and rapidly. Further, these changes are essentially unpredictable. The manager himself doesn't know what changes will occur. If he did, he could set up the procedures in advance and delegate the decision process to a subordinate. It would no longer be a managerial concern. He might continue to be concerned with how well a problem was being handled, but he would not be directly concerned with the problem itself.

In other words, as soon as a problem becomes predictable and the method of handling it can be specified, the manager, functioning as a manager, moves his attention to a higher level and becomes concerned with a more global view of the operation. This higher-level consideration remains his concern precisely because, at that level, there are unpredictable factors. So I take it as inherent in the managerial situation that the manager cannot predict what problems he is going to be dealing with or what information he will need to deal adequately with them. He responds to very subtle effects within the organization and within the world in which the organization is acting. Furthermore, his responses are to a large extent influenced by his own internal state of being, in that they depend upon his experience and his awareness of where there has been previous difficulty and success. So the manager is himself in a learning process and therefore adds a further unpredictable factor to the situation. Since the success

of the top-down method of approach depends to a large extent on the ability to predict future needs and use, it is inherently limited as to what it can achieve.

Principle-Oriented Approach

As a result of these considerations it occurred to us that it was necessary to find something in between these two general methods of approach. The approach we have found we sometimes call "principle-oriented." We argue that the system should be based on principles that are derived from top-down considerations. However, the actual implementation should include adaptation and evolution in such a way that it will come at least reasonably close to tracking the use of the system in the immediate environment. We can say, then, that the system will exist at any moment in a way that expresses a bottom-up kind of responsiveness.

One of the first principles we see, therefore, is that the system should have the ability to adapt. Adaptation is an old word that has been applied in many areas of computer design and use. In a sense, any time a one-time memory assignment is done, there is adaptation. Any time the system includes a "paging" procedure or "virtual memory" structure, there is adaptation. I am using the word here in a more global context. Examples of what I mean include consideration of what procedures should be available in the system, what kind of searches of the data base should be available, and what procedures should be available for processing and outputting the result of such searches. The answers to these questions may change with time as managerial attention shifts. If the system is capable of tracking these changes, it will be adaptive at a level that involves the whole state of the system and the nature of its response to a given interrogation.

The adaptation referred to here is that in which the system causes a change in its own configuration. This change may occur in any level from the specific implementation of a particular file structure to the choice of which files are maintained and where. It may include a change in the data structure type used in a file. The result will be to change quite drastically the procedures with which the system can respond to an interrogation. What we seek, then, is a set of concepts and principles that provide a general guide to information-processing systems design for supporting truly managerial functions.

This line of thought led to the general schema shown in Fig. 11-1. At the top level is the box labeled Purpose, which is the motivation for the system. At the next level is the Nonadaptative Properties of the system. These include those aspects of the system that provide the frame within which adaptation occurs. Most obviously, the hardware is nonadaptative. It may be evolutionary in the sense that it can be changed from the outside, and it probably

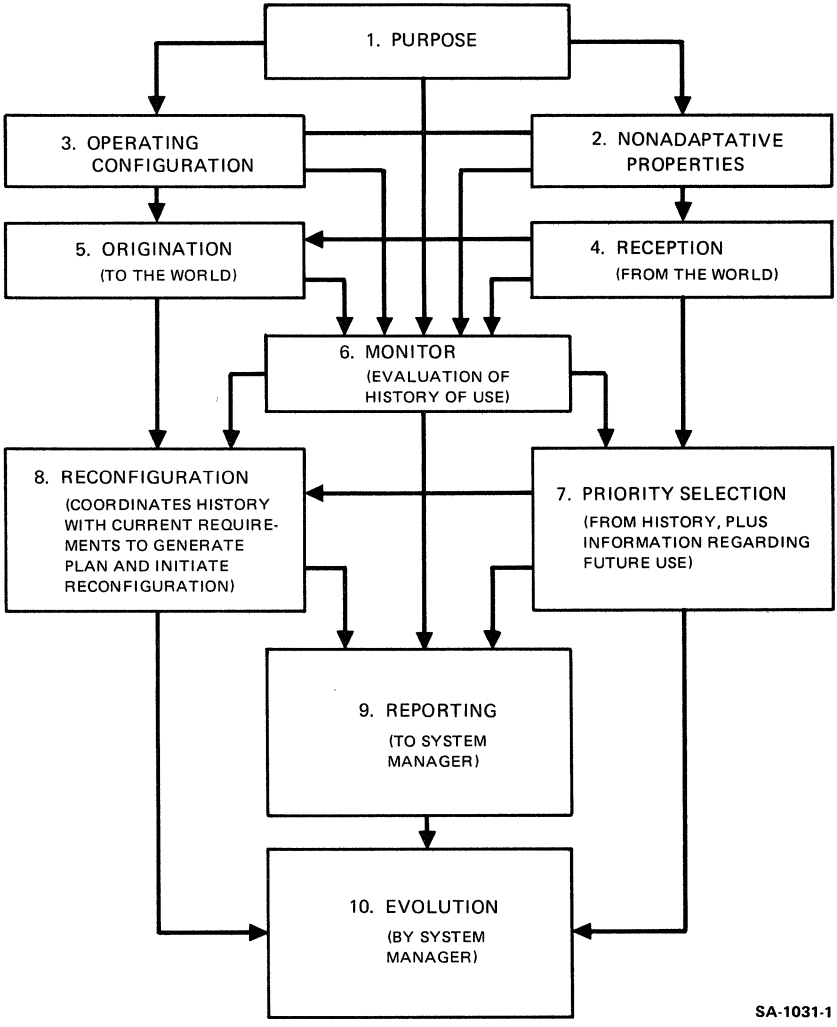


Fig. 11-1. Schema for an Adaptative and Evolutionary Management Support System

will be as time progresses. But at any given instant the hardware will be given. However, much more than the hardware is not subject to adaptation. We must, for example, have either a fixed format in the basic data file or at least some fixed rules for interpreting whatever is there. At some level the interpretation of data files must be nonadaptative. Another area is the interrogation language. It must either be nonadaptative or depend upon some metalanguage that is nonadaptative. In general, almost every function of the system must, at some level, have a nonadaptative base of operations.

In the next box is what we call the Operating Configuration, which describes the actual state of the system at a given time. It seems a valid inference that we would like to decouple, as far as we can, the operating configuration from the nonadaptative properties. The further apart these are, the greater the range of adaptation that is possible and the greater the opportunity for tracking changes in the environment of use. On the other hand, the further apart these two are, the harder it is likely to be to keep the system in a stable functioning state. This is really the crux of the problem: how to separate adaptativity from the nonadaptative base so as to achieve the greatest possible range of adaptative response without having the system become unstable or nonresponsive.

At the next level is the box labeled Reception, the means by which the system will receive from the outer world. It will include means for inputting and updating data and inputting any commands, interrogations, or any other communications from the outside world. The corresponding box on the other side is labeled Origination. This includes means of processing data in response to an interrogation and developing the reply to an interrogation. It will include the actual process of communicating to the outside world. This function will also include the issuance of any routine reports that are independent of any actual command. It may also include monitoring the data to provide automatic warning when the data suggest that something deserves the attention of the manager. Origination may function in response to either a reception or an internally generated command.

So far we have not introduced any adaptation. The next two levels are concerned with this aspect. The first of these levels has the single box labeled Monitor. This has the function of keeping track of what the system is doing, including both its use of the various resources within the operating configuration and the resources it could have used if they had existed. To put it anthropormorphically, it will keep track of where the system is hurting and where it feels comfortable. This record will provide the system with evidence as to what its environment of use has been. To the extent that adaptation is based on extrapolation of past use, this record will be the major source of the decision as to what adaptation should be executed.

In some cases, the system may have additional information as to probable future use given either incidentally during inputted interrogations or as specific predictions from the user. Such information will be received through Reception. Priority Selection will integrate the information derived from the Monitor with this additional directly inputted information. To establish a goal for adaptation, it will establish what the configuration of the system should be on the basis of the existing information available to the system.

The determination of the target configuration for adaptation will still not be sufficient. This target must be integrated with the ongoing operations of the system. It must also be modified according to the cost of executing the desired adaptation and the value that is expected to accrue from it. These processes will be performed in the box labeled Reconfiguration, which will serve as the executive for the actual adaptation process itself.

The need for the evolutionary process will remain. By this is meant the changes introduced by command from an external source, which we call the system manager. Evolution must come into play when the scope of the system is to be enlarged or when new facilities are to be added. It may also come into play because of a sudden drastic shift in the external environment; for example, the occurrence of an international crisis. This might dictate a sudden shift in the environment of use, which could be anticipated by the system manager but which would not be recognized quickly enough by the adaptative procedure. In such a circumstance, the system manager might be required to intervene, to cause a change in the operating configuration by direct command. More frequently the system manager will be concerned with tuning up and expanding and improving the adaptation process itself. He will seek to improve the prediction algorithm, and perhaps the priority selection algorithms, to improve the adaptation process. The ideal use of the evolutionary capability, as we see it, would be analogous to a higher level adaptation rather than a substitute for adaptation.

Given the general scheme, the next step is to expand it to something that will begin to describe a possible system. This has led to the functional block diagram shown in Fig. 11-2.

Figure 11-2 is laid out to more or less correspond to the schema of Fig. 11-1. The correspondence is not exact but does follow through fairly well. (Also, there are possible variations in this functional block.) My intent is to show that the system concept is feasible and to indicate the main problem in which research is needed.

Corresponding to the nonadaptative properties is the nonadaptative file system. Its primary contents will be the raw data in updated form. Although some adaptation may be possible within these data files, this must be done with great care. In an adaptative system, the user does not know the exact current state of the

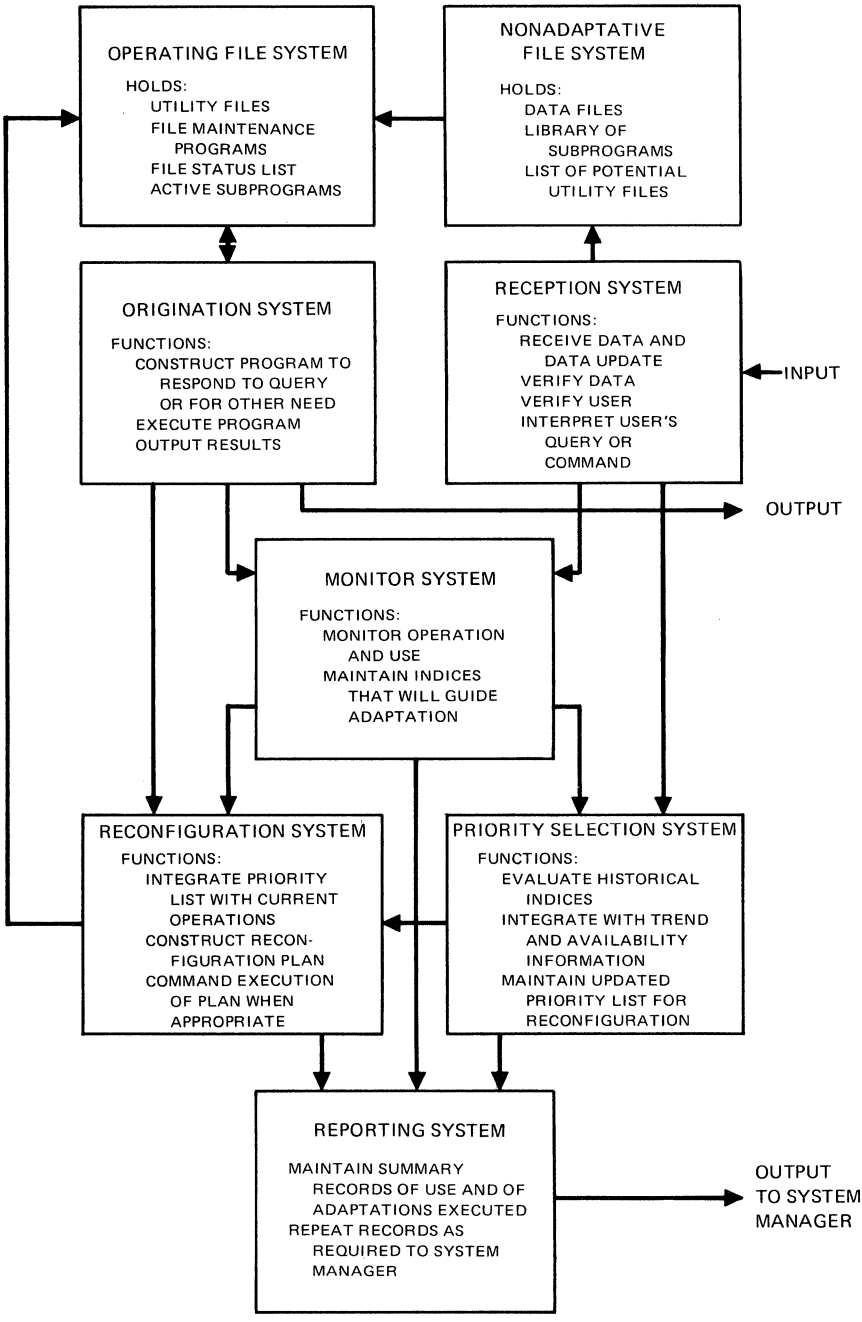


Fig. 11-2. Proposed Management Support System

system, yet he must be able, as a last resort, to refer his questions back to the raw data without ambiguity. Further, the main data files are presumed to be very large, perhaps being contained on many tapes. A change requiring the reprocessing of the data files will therefore be very expensive and should be undertaken only to accrue a substantial benefit. Generally, data files should not be subject to adaptation.

Corresponding to the Operating Configuration is the Operating File system. The main content of this box is termed the Utility Files. These files will be derived from the main data files and will, it is hoped, be sufficient to handle most of the interrogations of the system. We visualize the utilities files as being sufficiently small to be held within the system, at least on the disc memory. Our intention is that most of the processing done by the system will be through the utility files, with the data files being referred to only in a default situation where the utility files are insufficient. For this purpose, the utility file system will contain various compilations and samplings, and whatever else seems to be appropriate to the expected environment of use. The immediate area of adaptation will then be on the questions of what utility files are to be maintained and how often they are to be brought up to date. In addition, there may be adaptation of the actual structures of some utility files. There are file structures, some of which we have studied, whose structures can be manipulated adaptatively without excessive difficulty.

The Reception system will receive new data, update of old data, and validate data. Data validation is an exceedingly important problem in any practical system. The degree to which incorrect data are screened out of the system will have a great deal to do with the manager's acceptance of the system. To anticipate, this is one identified area of research we feel to be necessary.

The Reception system will also verify the user's right of access. This will involve security and privacy considerations. This function is important, even though we do not consider it intrinsic to the research purposes of the program.

The Reception system will also interpret the user's interrogation. This will involve more than simply providing an interrogation language. Since the operating configuration is separated from the user, he will not know exactly what utility files are available or how current they are. He must, therefore, be able to enter his interrogation in a way that is independent of the operating configuration and that does not specify the precise procedure to be used in response. This, then, imposes quite different restraints on the interrogation language from those that are usually involved in the interface between computer and user.

The block labeled Origination system will have the task of responding to interrogations or, as appropriate, to an internally perceived need (if an automatic warning capability is included).

Because the configuration that the system is in at the moment will not be visible to the user, the Origination system must be able to accept the interrogation and translate it into a program that will function effectively with the existing utility files. For example, a particular interrogation may be easily handled by using certain utility files. These files, however, may not be fully up to date. Beyond their date it may be necessary to use a default procedure having recourse to the primary data files. The Origination system must determine a strategy appropriate to the existing state of the system and respond accordingly.

Coming to the Monitor system and the adaptation processes, we expect to keep track primarily of the use of the utility files and, perhaps, of the various subprograms. As we now visualize the functioning of the Origination system in response to an interrogation, it will construct a strategy or a program that will be responsive to the interrogation. This program will be checked against the existing utility files to determine that it is operable. If it is, Originations will execute it and pass to the Monitor system the information that these utility files have, indeed, been used. If the program is not executable because the required utility files are lacking or not sufficiently up to date, this information will be passed to the Monitor system, and the Origination system will then construct a second-best program. This process will continue until a program is found that will run, in the worst case, by default to the data files themselves. The Monitor system will then receive information about which files were used and also about which files were sought but not found usable.

The record of use and unavailability-for-use will be passed from the Monitor system to the Priority system, where it will be combined with any other information about probable future use. It will also be coordinated with the cost functions that determine the costs of making various changes to the Utility Files. The Priority Selection system will compile a priority list of changes. This, in turn, will be passed to the Reconfiguration system that integrates the priority list with current operations and constructs a reconfiguration plan. The Reconfiguration system will also command the execution of this plan when it is appropriate in terms of the current usage of the system.

Finally, the Reporting system will output the information to the system manager for use in guiding system evolution. This system will obtain its information primarily from the Monitor, Priority Selection, and Reconfiguration systems.

The above gives some idea of the SRI system concept. Considerable elaboration of the concept is available in the First Technical Report on our contract, which is available to those who are interested. The main purpose of this analysis, however, has been to identify the areas of research that are urgently needed for this type of system. In general, we have identified seven such areas.

RESEARCH AREAS

The first area of research identified is the adaptative process itself. Much analysis has been done on the general concept, but we do need to obtain direct experience of such a process in order to determine what really will make it go. Therefore, one of the primary areas of research will be to set up a data base in which we can play adaptative games on an effective and realistic level. The data base chosen is being drawn from the Navy 3M system (Maintenance and Material Management). The 3M system receives a report of every maintenance operation performed in the Navy, as well as collateral inputs; it is, therefore, a huge data base. We are acquiring a small subset of it for use in our system, which may be extended as time goes on. The scenarios we intend to construct on it initially are concerned with setting up and monitoring operational standards.

The second main area of research will be in data file structures themselves. Much work has gone into file structures for a large number of systems, and considerable experience has been gained with them. However, a significant difference of emphasis is imposed by our particular use of these files. In general, a file structure usually is chosen to facilitate some particular uses. Here, however, the primary use of the data files will be to construct the utility files. Since this can be done when the system is not otherwise occupied, the efficiency of this operation will not be of major importance. The primary concern is to ensure that no information is lost. We shall also need to be sure we can input data revisions with relative convenience and with the least interruption of system operation. In our data files, the efficiency of use must take a much lower priory than the efficiency of adding to or modifying the resident information. This will put quite a different condition on file structure than is normally the case. Hence, research is needed to determine the implications of this shift in emphasis.

The third area of research will be the actual generation of the response in the Origination system. How is the response to be generated from the input interrogation and current knowledge of the system? This is vital, because we want the system itself to be essentially transparent to the user. The user, particularly if he is a high-level manager, is not going to worry about just which utility files are up and which are unavailable. The system itself must make this connection and put the appropriate pieces together. A related question in this area of research has to do with the capabilities of the query language being used. To illustrate, consider the use of what may be called indefinite qualifiers. This includes words like much or many or generally, or for example, the statement: A is usually followed by B. What does this usually mean in this context? Such words are not normally part of the programming language; yet they seem to be inherent in the thought processes to

which we are trying to respond. Probably the most critical period in the managerial decision process is when the manager is trying to pin down exactly what it is that he should be concerned with. He is, perhaps, groping towards some possibility that he only dimly senses. At such times, he can describe his notions only in intentionally vague terms. I am not convinced of the importance of a true natural language capability, but I strongly suspect the system we hope to achieve here will have to reflect modes of thought that are possible with a natural language.

The fourth area of research concerns the utility files. What should be their structures? Are they to be adaptative in themselves and, if so, how? There are possibilities for self-adaptative file structures, some of which we have begun to study. What are the implications of these various possibilities? How should we choose between different possible structures for the utility files? Another question involved here is that implied in the statement that any utility file is in some sense a compressed version of the data file. It is compressed because it is much smaller and also entails considerable information loss. Different kinds of compression entail different kinds of information loss. There is a need to understand the tradeoffs involved and to explicate the principles upon which the necessary decisions should be made.

The fifth area of research concerns interactive processes for hypothesis generation and test. This is really at the heart of modeling problems. It is also at the heart of making a truly creative synthesis of the man and the system in terms of model building. The extent to which the system can accept vaguely worded, probing types of hypotheses will largely determine the subtlety of the interaction obtainable. The system that has subtle response capabilities in this area will permit the user to feel his way into his problems. It will generate the kind of mutual creativity we would like to achieve.

The sixth area of research is data verification and validation. Data validation is seen as having three aspects. The first concerns the collection of the data and its input into the system. This aspect is outside the system itself and, although it is a key operational area, it is not an area of immediate concern to us. The second aspect is validation on entry. This concerns the question of whether the data are properly formatted and whether the values are reasonable or otherwise not obviously wrong. At its most immediate level, this aspect involves checking for format, for self-consistency, and against various predetermined tests for possibility or plausibility. At a higher level, the plausibility tests themselves can be adjusted adaptatively as experience is acquired. This leads to some interesting questions of strategy and of the measures of value and cost that should be used.

The third aspect of data validation concerns the possibility of cross-checking data already within the system to determine whether they are consistent with a possible or probable model of the world.

This aspect is very closely related to the problem of hypothesis generation and test; the difference, perhaps, is that here we are talking about hypotheses that are internally generated and that are confined to a much smaller range of possibilities.

The seventh and final area of research identified is output generation. It is recognized that the ability of the manager to perceive relationships among data is highly dependent on the precise way the data are presented to him. The presentation of the outputted information is very important to making the system an integral part of the decision-making process. Consequently, great care must be taken in how we exploit the capability of various kinds and modes of data presentation. I do not refer to the specific hardware that should or should not be used, but to the manner in which that hardware is used, including, for example, whether the data are presented as graphs, tables, movies, or what. Research is needed to identify the appropriate means to use under different conditions.

SUMMARY

This completes the survey of our work on large file management systems. We are at the beginning of what we hope will be a continuing research effort. Therefore, the work so far has been mainly concerned with determining the directions to take. At some future point, I hope to report specific results on the several problems delineated here.

PART III. RESEARCH PROJECT REPORTS

The ten chapters in this section are representative of the research, development, and survey projects currently under way in the multi-access area. They cover a broad area of applications, including potential consumer, industry, and government users. The chapters have been divided into three general categories of software, hardware, and man/machine systems. The separation is, however, quite arbitrary and most papers cover material in all three areas.

SYSTEMS SOFTWARE

This category includes three papers whose primary emphasis is on data management systems and operating systems.

Dr. Wier presents Bell Labs' experience in developing data management and retrieval systems and discusses some of the data-based applications they have in operation. Dr. Corbato presents the history of the Multics System at Project MAC of MIT. Dr. Liskov then discusses the microprogramming and software involved in the Venus Operating System.

PROCESSING SYSTEMS

This category contains three papers whose emphasis is computer or terminal hardware.

Mr. Volk presents an interactive system utilizing cable TV that is being tested for such applications as computer aided instruction. Dr. Bell discusses performance evaluation before and after implementation, using a graphics terminal as an illustration. Mr. Weitzman then presents the results of a recent survey and forecast of computer hardware performance and capabilities.

MAN/MACHINE INTERACTION

This category contains four papers whose emphasis is on man/machine dialogue or machine augmentation of human capabilities.

Mr. Bernstein discusses three projects leading toward a more natural man/machine dialogue; a natural English data retrieval system, a speech understanding system, and a data tablet graphic and hard printing input system. Dr. Dostert presents the Rapidly Extensible Language (REL) System being developed at Cal Tech. It offers the ability to dynamically tailor an interactive language through definition of terms and interrelationships. Dr. Goodenough discusses a computer-directed training system that is oriented toward on-the-job training. Dr. Dixon then closes the section with a paper on some of the analysis systems being developed and used in the health sciences area at UCLA's Health Services Computing Facility.

OTHER CURRENT RESEARCH PROJECTS

The ten papers included in this section are only a small cross-section of the research being performed today under government sponsorship and do not encompass the wide varieties of multi-access computing research currently under study.

To obtain a broader cross-section of current research endeavors, one need only observe some of the specific activities being funded by the Information Processing Branch of ARPA, the Advanced Research Projects Agency of the Department of Defense. A detailed accounting of the individual research projects would fill volumes. Thus, only an indication of some of the key projects at each of 22 ARPA funded labs is provided on the following pages.

1. University of California, Berkeley

   New computer architectures-studies based on a highly modular structure employing a number of CPU's memory modules, discs, etc.

2. Bolt, Beranek and Newman, Inc.

   Natural Communication with Computers – Application of a semantic network memory; development of an extensive augmented, transition net grammar of English for data retrieval; further development of LISP for PDP-10; and development of the TENEX time sharing operative system for PDP-10.

HARDWARE DESIGNS FOR ARPA NETWORK IMPS

3. Harvard University

   The ARPA Network – Hardware interfaces between PDP-10 and IMP; software contributions to network protocol.

   Computer Graphics – Introduction of remote computation facilities in aid of graphics.

4. Washington University, Computer Systems Laboratory

   Molecular Graphics – Using a computer system to manipulate and display models of molecular graphs.

   Macromodular Systems – Establishment of a pilot inventory of electronic computer units.

5. Stanford Research Institute, Augmentation Research Center (ARC)

Network Information Center – Will provide online network information services for the ARPA network.

Dialogue Support System – Provides means for accumulating, retrieving, and studying the communications generated with ARC.

6. University of Utah

Waveform Processing – Use of computers to process signals (both pictures and sound).

Graphics – Development of economical graphic communication systems useful in human tasks.

Computing Structures – Real-time experimentation with novel computing systems and components.

7. Stanford University, Heuristic Dendral Project

Artificial Intelligence – Focused on understanding the processes of scientific inference in physical chemistry.

8. MIT, Lincoln Laboratory

Graphics – Development of graphic man-machine communication techniques.

9. Network Analysis Corporation

Network – Analysis and design of the ARPA network; studies of properties of large computation-communication networks.

10. Case Western Reserve University, Project LOGOS

Security – Creation of a design tool to produce systems that are certifiable as being secure.

11. University of California, Santa Barbara

Networks – Online software systems developments for the ARPA network.

Speech – Research effort into techniques for voice input.

12. Applied Data Research, Inc.

Research in machine-independent software programming.

13. UCLA

    Networks – Specific areas under investigation include network software, network measurements, and computer systems modeling and analysis.

14. Dartmouth College

    Time-shared Computing Systems – Projects include continuing development of the Dartmouth Time Sharing System; graphics display devices, and applications in library automation and radiology treatment.

15. MIT, Project MAC

    See Chapter 13 for a complete report on MULTICS.

16. Computer Corporation of America

    Investigations of data handling in computer networks.

17. University of Illinois

    ILLIAC TV – Total hardware and software systems development.

18. MIT, Artificial Intelligence Group

    Robotics – Development of high-level hand-eye system for economic planning decisions. An associated project is to develop an English semantics program appropriate to robotic environment.

    PLANNER Language – New programming language important in solving problems.

    Mathlab – Continued development of a facility for automated algebraic manipulation.

19. Carnegie – Mellon University

    Artificial Intelligence – Variety of studies in programming languages, automated theorem proving, semantic networks, chess and protocol analysis.

    Hardware – Studies on the design of register transfer modules for digital systems designs.

20. Stanford Research Institute, AI Group and Computer Science Group.

Artificial Intelligence – Various studies in goal seeking and problem solving methodologies in robotry, speech understanding, and in industrial applications.

Large File Management Research – A study program seeking to understand how informational systems can be made responsive to the needs of high level managers.

21. RAND

Climate Dynamics – Focused on the problem of climate determination and control.

Networks – Development of programs to utilize remote resources.

Adaptative Communication – Attempt to make the man-machine interface more flexible and man-oriented.

Computer Program Organization – Study to provide the interpreter of problem oriented languages with judgment and problem solving power.

Computer applications to the military – Enable the military to more effectively utilize computer research.

22. System Development Corporation

Computation-Communication Tradeoff Studies – Design to investigate a possible mismatch between DoD computation and communication requirements in the 1975 to 1980 era.

Natural Languages – The CONVERSE system is concerned with constructing and querying online data bases in natural English.

Graphic I/O – Development of interactive computer graphics capabilities.

Voice I/O – Development of man-machine interaction systems using continuous speech.

Networks – ARPA network protocol research effort and application of graph theory to compiler organization.

Security – Provide practical security controls in multi-access systems.

Information on any of these projects can be obtained by writing directly to the research lab of interest.

12. Interactive Information Systems

Joseph M. Wier

Bell Telephone Laboratories

Holmdel, New Jersey

We've been in what I call the information management business for seven years. We began by undertaking a number of studies of the switching requirements of the Bell System. In each case, the studies involved a relatively large mass of descriptive data characterizing the switching plant and its environment. Typically, it seemed continually necessary to return to the basic information for different or more precise results.

Programming a new look at the data base using standard batch processing techniques was tried at first. Later, precomputed summaries were attempted which hopefully would answer the more important questions. The tailored program approach was adequate for straightforward questions with no time pressure. The precomputed summaries worked for questions lending themselves to preplanned computations, but the data produced was an unmanageable four foot pile of computer printout.

Both efforts strongly suggested the desirability of more immediate and flexible data access. Thus, the results have been a sequence of information systems, each built on knowledge gained from previous systems and tied to some application or applications. The early systems were "hard-wired" information retrieval-only systems. Later ones included both data retrieval and data entry, extensive administration capability and vastly improved levels of speed, modularity of design and portability.

In evolving, the systems began to assume a form dictated by their applications. The following characteristics are perhaps worth noting:

1. The data bases were hierarchically oriented.

2. The systems were built on time-shared computer operating systems.

3. Most of the systems were used by large numbers of people distributed over large geographical areas.

4. Most of the users or potential users were not programmers.

5. Many of the users were possessed of a low level of technical training.

Because broad system access was necessary, commercial time-sharing services were used. To meet the needs of a broad class of users, the basic complexities of the systems are hidden behind a simple facade and responsibility for keeping the user out of trouble and the system protected is vested in the information system. These requirements resulted in demands placed on the time-sharing systems that could not always be conveniently met, including requirements for:

1. Relatively large core storage (>128K bytes).

2. Large random access secondary storage (tens of millions of bytes).

3. Programmer control of file access.

4. A time-sharing accounting system allowing extensive user interaction but individual charging.

5. Extensive program and file sharing capabilities with programmer control of these interactions.

6. A computational capability large enough to support many users with rapid response.

The systems have all evolved toward a specific modular form. The current system is split into roughly four packages.

1. An administrative package for taking care of the endless details associated with collecting, changing and monitoring data base performance.

2. A "front end" language package allowing the user to specify his own user and job oriented language and to implement it without appreciable delay. It also serves to parse the resulting language and to supply suitable information at a standard internal interface to serve other system packages.

3. A processor package for composing internal calls on a data management system to store, change, and retrieve from the data base and for returning information to the user, if required.

4. A data management system for organizing, adding to, retrieving from and administering hierarchically organized data bases.

These packages will not be discussed in any detail, but the latter three are characterized by the applications leading to them.

The languages were generated by a compiler-compiler, which prepares tables describing the language in syntactic and semantic context from a modified BNF description. These tables are used by a run-time system for executing user-machine conversation. The system generally uses a keyword-oriented, phrase structured grammar. It is readily understood, but no attempt is made to make it totally free, even in appearance, because normal English is too ambiguous. We thus sacrificed appearance for accuracy. After years of use by non-programmers, we are not sorry. However, as our users tend to need rather straightforward results, most of the processing is simple in concept, although requiring a good deal of computation. It is possible to produce processors independent of the organization, size and content of the data bases used. Because of the interactive nature of queries, additions and changes made to the data, it was desirable to limit the complexity of executed requests so that the user maintains a running intuitive understanding of the process he is controlling. By so doing, the user feels that he is part of the process and actually contributes materially by recognizing significant results and choosing useful directions for later interaction.

The data management system separates the description of the structure of the data base from the storage of the data. It manages the allocation, storage, and retrieval of information and directs data base computations. To service numerous users out of the same data base, it allows for concurrent actions by many users. The system is designed to work in an interactive environment where change is frequent. The hierarchical structure of our data bases is mirrored in the method of managing the data. In some cases the hierarchical bias will make the system less useful for other forms of data, but the sacrifice will not show in our applications.

To further circumscribe the types of systems we have built, let us look at some simple requests which show the character of an interaction. The following are representative of the system although the data bases implied are fictional.

```
PRINT EMPLOYEE NAME, SOCIAL
SECURITY NO.: IN NY COMPANIES:
WHEN AGE = 63: GO:
```

TOTAL SALARIES: IN COLORADO:
WHEN TOWN WORKED = DENVER AND
CLASSIFICATION = SALESMAN: GO:

All of the usual synonyming, order interchanging, multiple syntaxing, diagnostics, and aids go along with the system. In addition, virtual data base elements can be added by defining new elements as functions of those stored in the data base. An example is:

LET PERF RATIO = SALARY/AVG SALARY

The new element, PERF RATIO, can be used as though it were in the data base. It will merely be computed as needed from items already contained in the data base, in this instance SALARY and AVG SALARY.

It is characteristic of the applications we have serviced that the users are spread over the Bell System. This situation shapes the form of our interactions as these users largely employ teletypewriting terminals made by the Teletype Corporation. Thus, the systems we design anticipate this environment and so no special or sophisticated terminals are serviced. This rules out all light-pen cathode-ray-tube operations and limits graphical output to very simple teletypewriter types. Similarly, higher speed terminals are also not serviced and so any transactions involving higher printing speeds are also ruled out. This further simplifies the individual transaction.

Some of the things we have learned are probably typical only of our applications. Most of them we feel to be rather more general than that as they follow from the behavior of people rather than from the specific requirements of the jobs we have carried out.

The first and probably most important observation concerns the system user. He is both our worst enemy and our greatest ally. He is our worst enemy because he, by indirection or conscious effort, tends to poor memory, sloppy work habits, careless reading of manuals, and incomplete knowledge of the information stored. He thus will misuse and misunderstand any system that is not carefully protected from such treatment. On the other hand, he has some capabilities not found in even the most ingenious of computer systems. He has better knowledge of his job and situation than is available from any other source; he is an extremely talented pattern recognizer, who can detect significance or regularities with astonishing ease; and only he will have the capability to act and to judge on any action. In such roles, he is an indispensible ally and allows the combined user and information system to do things which would not be possible for either alone.

To service him, it is necessary to remember that he has the usual human failings and to design the system to accommodate these. Furthermore, since it is better he be an ally than an enemy, it is

wise to help him in his efforts and to ignore his lapses. Thus, our system comes equipped with many peripheral aids such as diagnostics, which are not critical of the user, teaching aids, catalogs, initializing processes and system trapping of abnormalities. In general, the system should be helpful under all conditions of adversity. It should anticipate the many unusual difficulties the user can precipitate and help him out of them.

A second important lesson is that the user is the standard to which the system is tuned. If it fails, the system is at fault, not the user. To ignore this principle is to be ignored by the user at best and to be sabotaged by him at worst.

A third thing we have learned is that the system should not bore the user. Thus, the system must produce reasonable response time and should do something for him that he needs done. Not only does poor service annoy the user; it also handicaps him in solving his particular problem. The considerable lapses in time between interactions can take his mind off the job he is doing and can lower his efficiency in using the data he gets from the system. The user and information system work together to deal with the problem on the user's mind. Prompt response holds off boredom and demands less of the user's somewhat fallible memory. It also tempts the user to apply his talents in pattern recognition and decision making more frequently and that, in itself, often speeds the information system along the path it must follow.

In each of the first three points, it cannot be overemphasized that the user is not the fool that computer people sometimes paint him. He happens to be good at some things needed to solve the problems at hand. Fortunately, the user's weaknesses are the information system's strengths. Thus, they work well together.

A fourth lesson thrust upon us is that good systems can be designed only in the presence of the potential users and applications. By using the system in numerous applications and assiduously watching its performance, it gradually can be adapted to the needs of the application and the user. Ingenious ivory tower solutions frequently do not work or they solve problems that don't exist.

A fifth principle is that things should be kept simple for the user. The more complex the system, the more likely it is that it won't get used or will be misunderstood or misused. Thus, it should be easy to sign on to the system; it should be easy to learn to use it; it should not change rapidly or without adequate warning; it should provide results which are easily absorbed in one "mindful"; that is, the results should fit intuitively into the user's picture of things; it should present data in a form easily absorbed by the user; it should use his responses to help it get results, which implies that it should use simple user-oriented instructions; it should enable the user to make simple checks or experiments in a way natural to him.

Sixth, the system should do something for the user, something he wants or potentially needs done. To achieve this end, it is necessary to listen to the users, to look for their criticisms, to study their uses, and to keep a journal file to watch it in operation. Most important, it is necessary to look for signs of disenchantment, most notable by a reduction in use.

Finally, the total problem will not be solved quickly. The systems won't fit all potential applications functionally; it will be too costly in some cases; the terminals won't be available in others; the state of the art won't allow an adequate solution in many instances; and finally, users may not be ready for it. Thus one conclusion may be that abandonment is in order in some applications.

We have attempted to apply these observations to the design of our existing system. It has achieved relatively wide acceptance. It is not at the state we would like to see it. It is somewhat more complex to use than we would like. It costs too much to store information. The access costs are too high for many systems. We still occasionally find users who get into difficulties. We are not entirely happy with the user terminal situation. However, it does operate. It does service nonprogramming users. It does aid a user to understand complex data bases. It can handle enough information to meet many important applications. Further, it can service enough users so that the data base serves as a real coordination point for the situation mirrored there.

13. Multics: The First Seven Years*

F. J. Corbato
Massachusetts Institute of Technology
Cambridge, Massachusetts

C. T. Clingen
Honeywell

J. H. Saltzer
Massachusetts Institute of Technology
Cambridge, Massachusetts

In 1964, following implementation of the Compatible Time-Sharing System (CTSS)[1, 2], serious planning began on development of a new computer system specifically organized as a prototype of a computer utility. The plans and aspirations for this system, called Multics (Multiplexed Information and Computing Service) were described in a set of six papers presented at the 1965 Fall Joint Computer Conference[3, 8]. The development of the system was undertaken as a cooperative effort involving Bell Telephone Laboratories (from 1965 to 1969), the computer department of the General Electric Company, ** and Project MAC of MIT.

Implicit in the 1965 papers was the expectation that there should be a later examination of the development effort. From the present vantage point, however, it is clear that a definitive examination cannot be presented in a single chapter. As a result, the present chapter discusses only some of the many possible topics. First we review the goals, history, and current status of the Multics project. This review is followed by a brief description of the appearance of the Multics system to its various classes of users. Finally, several topics are given which represent some of the research insights which have come out of the development activities.

This organization has been chosen in order to emphasize those aspects of software systems having the goals of a computer utility which we feel to be of special interest. We do not attempt

---

*This paper is a reprint of the paper entitled "Multics: The First Seven Years" by F. J. Corbató, J. H. Saltzer, and C. T. Clingen presented at the 1972 Spring Joint Computer Conference.

**Subsequently acquired by Honeywell Information Systems, Inc.

detailed discussion of the organization of Multics; that is the purpose of specialized technical books and papers.*

GOALS

The goals of the computer utility, although stated at length in the 1965 papers, deserve a brief review. By a computer utility it was meant that one had a community computer facility with:

1. Convenient remote terminal access as the normal mode of system usage;
2. A view of continuous operation analogous to that of the electric power and telephone companies;
3. A wide range of capacity to allow growth or contraction without either system or user reorganization;
4. An internal file system so reliable that users trust their only copy of programs and data to be stored in it;
5. Sufficient control of access to allow selective sharing of information;
6. The ability to structure hierarchically both the logical storage of information and the administration of the system;
7. The capability of serving large and small users without inefficiency to either;
8. The ability to support different programming environments and human interfaces within a single system;
9. The flexibility and generality of system organization required for evolution through successive waves of technological improvements and the inevitable growth of user expectations.

In an absolute sense the above goals are extremely difficult to achieve. Nevertheless, it is our belief that Multics, as it now exists, has made substantial progress towards achieving each of the nine goals.** Most important, none of these goals had to be compromised in any important way.

---

*For example, the essential mechanisms for much of the Multics system are given in books by Organick[9] and Watson[10].

**To the best of our knowledge, the only other attempt to comprehensively attack all of these goals simultaneously is the TSS/360 project at IBM[11, 12, 13].

HISTORY OF THE DEVELOPMENT

As previously mentioned, the Multics project got underway in Fall 1964. The computer equipment to be used was a modified General Electric 635, which was later named the 645. The most significant changes made were in the processor addressing and access control logic, where paging and segmentation were introduced. A completely new Generalized Input/Output Controller was designed and implemented to accommodate the varied needs of devices such as disks, tapes, and teletypewriters without presenting an excessive interrupt burden to the processors. To handle the expected paging traffic, a four-million word (36-bit) high-performance drum system with hardware queueing was developed. The design specifications for these items were completed by Fall 1965, and the equipment became available for software development in early 1967.

Software preparation underwent several phases. The first was the development and blocking out of major ideas, followed by the writing of detailed program module specifications. The resulting 3000 typewritten pages formed the Multics System Programmers' Manual and served as the starting point for all programming. Furthermore, the software designers were expected to implement their own designs. As a general policy PL/1 was used as the system programming language wherever possible to maximize lucidity and maintainability of the system[14, 15]. This policy also increased the effectiveness of system programmers by allowing each one to keep more of the system within his grasp.

The second major phase of software development, well underway by early 1967, was that of module implementation and unit checkout followed by merging into larger aggregates for integrated testing. Until then, most software and hardware difficulties had been anticipated on the basis of previous experience. But what gradually became apparent as the module integration continued was that there were gross discrepancies between actual and expected performance of the various logical execution paths throughout the software. The result was that an unanticipated phase of design iterations was necessary. These design iterations did not mean that major portions of the system were scrapped without being used. On the contrary, until their replacements could be implemented, often months later, they were crucially necessary to allow the testing and evaluation of the other portions of the system. The cause of the required redesigns was rarely "bad coding" since most of the system programmers were well above average ability. Moreover, the redesigns did not mean that the goals of the project were compromised. Rather, three recurrent phenomena were observed: (1) typically, specifications representing less important features were found to be introducing much of the complexity; (2) the initial choice of modularity and interfacing between modules was sometimes awkward, and (3) it

was rediscovered that the most important property of algorithms is simplicity rather than special mechanisms for unusual cases.*

The reason for bringing out in detail the above design iteration experience is that frequently the planning of large software projects still does not properly take the need for continuing iteration into account. And yet we believe that design iterations are a required activity on any large scale system which attempts to break new conceptual ground such that individual programmers cannot comprehend the entire system in detail. For when new ground is broken, it is usually impossible to deduce the consequent system behavior except by experimental operation. Simulation is not particularly effective when the system concepts and user behavior are new. Unfortunately one does not understand the system well enough to simplify it correctly and thereby obtain a manageable model which requires less effort to implement than the system itself. Instead one must develop a different view:

1. The initial program version of a module should be viewed only as the first complete specification of the module and should be subject to design review before being debugged or checked out.

2. Module design and implementation should be based upon an assumption of periodic evaluation, redesign, and evolution. In retrospect, the design iteration effect was apparent even in the development of the earlier Compatible Time-Sharing System (CTSS) when a second file system with many functional improvements turned out to have poor performance when initially installed. A hasty design iteration succeeded in rectifying the matter but the episode at the time was viewed as an anomaly perhaps due to inadequate technical review of individual programming efforts.

CURRENT STATUS

In spite of the unexpected design iteration phase, the Multics system became sufficiently effective by late 1968 to allow system programmers to use the system while still developing it. By October, 1969, the system was made available for general use on

---

*"In anything at all, perfection is finally attained not when there is no longer anything to add, but when there is no longer anything to take away . . ." Antoine de Saint-Exupery, Wind, Sand and Stars.

a "cost-recovery" charging basis similar to that used for other major computation facilities at MIT. Multics is now the most widely used time-sharing system at MIT, supporting a user community of some 500 registered subscribers. The system is currently operated for users 22 hours a day, seven days a week. For at least eight hours each day the system operates with two processors and three memory modules containing a total of 384k (k = 1024) 36-bit words. This configuration currently is rated at a capacity of about 55 fairly demanding users such that most trivial requests obtain response in one to five seconds. (Future design iterations are expected to increase the capacity rating.) Several times a day during the off-peak usage hours the system is dynamically reconfigured into two systems: a reduced capacity service system and an independent development system. The development system is used for testing those hardware and software changes which cannot be done under normal service operation.

The reliability of the round-the-clock system operation described above has been a matter of great concern, for in any online real time system the impact of mishaps is usually far more severe than in batch processing systems. In an online system, especially important considerations are:

1. the time required before the system is usable again following a mishap,
2. the extra precautions required for restoring possibly lost files, and
3. the psychological stress of breaking the interactive dialogue with users who were counting on system availability.

Because of the importance of these considerations, careful logs are kept of all Multics "crashes" (i. e., system service disruption for all active users) at MIT in order that analysis can reveal their causes. These analyses indicate currently an average of between one and two crashes per 24 hour day. These crashes have no single cause. Some are due to hardware failures, others to operator error and still others to software bugs introduced during the course of development. At the two other sites where Multics is operated, but where active system development does not take place, there have been almost no system failures traced to software.

Currently the Multics system, including compilers, commands, and subroutine libraries, consists of about 1500 modules, averaging roughly 200 lines of PL/1 apiece. These compile to produce some one million words of procedure code. Another measure of the system is the size of the resident supervisor, which is about 30k words of procedure and, for a 55 user load, about 35k words of data and buffer areas.

Because the system is so large, the most powerful maintenance tool available was chosen – the system itself. With all of the system modules stored online, it is easy to manipulate the many components of different versions of the system. Thus, it has been possible to maintain steadily for the last year or so a pace of installing five or ten new or modified system modules a day. Some three-quarters of these changes can be installed while the system is in operation. The remainder, pertaining to the central supervisor, are installed in batches once or twice a week. This online maintenance capability has proven indispensable to the rapid development and maintenance of Multics since it permits constant upgrading of the user interface without interrupting the service. We are just beginning to see instances of user-written applications which require this same capability so that the application users need not be interrupted while the software they are using is being modified.

The software effort which has been spent on Multics is difficult to estimate. Approximately 150 man-years were applied directly to design and system programming during the "development-only" period of Fig. 13-1. Since then we estimate that another 50 man-years have been devoted to improving and extending the system. But the actual cost of a single successful system is misleading, for if one starts afresh to build a similar system, one must compensate for the nonzero probability of failure.

System	Development Only	Development + Use	Use Only
CTSS	1960 – 1963	1963 – 1965	1965 – present
Multics	1964 – 1969	1969 – present	

Fig. 13-1. A Comparison of System Development and Use Periods of CTSS and Multics*

*The Multics development period is not significantly longer than that for CTSS despite the development of about 10 times as much code for Multics as for CTSS and a geographically distributed staff. Although reasons for this similarity in time span include the use of a higher-level programming language and a somewhat larger staff, the use of CTSS as a development tool for Multics was of pivotal importance.

HOW MULTICS APPEARS TO ITS USERS

Having reviewed the background of the project, we may now ask who the users of the Multics system are and what the facilities Multics provides mean to these users. Before answering, it is worth describing the generic user as "viewed" by Multics. Although from the system's point of view all users have the same general characteristics and interface with it uniformly, no single human interface represents the Multics machine. That machine is determined by each user's initial procedure coupled with those functions accessible to him. Thus, the potential exists to present each Multics user with a unique external interface.

However, Multics does provide a native internal program environment consisting of a stack-oriented, pure-procedure collection of PL/1 procedures imbedded in a segmented virtual memory containing all procedures and data stored online. The extent to which some, all, or none of this internal environment is visible to the various users is an administrative choice.

The implications of these two views – both the external interface and the internal programming environment – are discussed in terms of the following categories of users:

1. System programmers and user application programmers responsible for writing system and user software.

2. Administrative personnel responsible for the management of system resources and privileges.

3. The ultimate users of applications systems.

4. Operations and hardware maintenance personnel responsible, respectively, for running the machine room and maintaining the hardware.

Multics as Viewed by System and Subsystem Programmers

The machine presented to both the Multics system programmer and the application system programmer is the one with which we have the most experience; it is the raw material from which one constructs other environments. It is worth reemphasizing that the only differentiation between Multics system programmers and user programmers is embodied in the access control mechanism which determines what online information can be referenced; therefore, what are apparently two groups of users can be discussed as one.

Major interfaces presented to programmers on the Multics system can be classified as the program preparation and documentation facilities and the program execution and debugging environment. They will be touched upon briefly, in the order used for program preparation.

Program Preparation and Documentation

The facilities for program preparation on Multics are typical of those found on other time-sharing systems, with some shifts in emphasis. For example, programmers consider the file system sufficiently invulnerable to physical loss that is is used casually and routinely to save all information. Thus, the punched card has vanished from the work routine of Multics programmers and access to one's programs and the ability to work on them are provided by the closest terminal.

As another example, the full ASCII character set is employed in preparing programs, data, and documentation, thereby eliminating the need for multiple text editors, several varieties of text formatting and comparison programs, and multiple facilities for printing information both online and offline. This generalization of user interfaces facilitates the learning and subsequent use of the system by reducing the number of conventions which must be mastered.

Finally, because the PL/1 compiler is a large set of programs, considerable attention was given to shielding the user from the size of the compiler and to aiding him in mastering the complexities of the language. As in many other time-sharing systems, the compiler is invoked by issuing a simple command line from a terminal exactly as for the less ambitious commands. No knowledge is required of the user regarding the various phases of compilation, temporary files required, and optional capabilities for the specialist: explanatory "sermons" diagnosing syntactic errors are delivered to the terminal to effect a self-teaching session during each compilation. To the programmer, the PL/1 compiler is just another command.

Program Execution Environment

Another set of interfaces is embodied in the implementation environment seen by PL/1 programmers. This environment consists of a directly addressable virtual memory containing the entire hierarchy of online information, a dynamic linking facility which searches this hierarchy to bind procedure references, a device-independent input/output[16] system, * and program debugging and metering facilities. These facilities enjoy a symbiotic relationship with the PL/1 procedure environment used both to implement them and to implement user facilities co-existing with them. Of major significance is that the natural internal environment provided and required by the system is exactly that environment

*The Michigan Terminal System[17] has a similar device-independent input/output system.

expected by PL/1 procedures. For example, PL/1 pointer variables, call and return statements, conditions, and static and automatic storage all correspond directly to mechanisms provided in the internal environment. Consequently, the system supports PL/1 code as a matter of course.

The main effect of the combination of these features is to permit the implementer to spend his time concentrating on the logic of his problem; for the most part he is freed from the usual mechanical problems of storage management and overlays, input/output device quirks, and machine-dependent features.

Some Implementation Experience

The Multics team began to be much more productive once the Multics system became useful for software development. A few cases are worth citing to illustrate the effectiveness of the implementation environment. A good example is the current PL/1 compiler, which is the third one to be implemented for the project, and which consists of some 250 procedures and about 125k words of object code. Four people implemented this compiler in two years, from start to first general use. The first version of the Multics program debugging system, composed of more than 3000 lines of source code, was usable after one person spent some six months of nights and weekends "bootlegging" its implementation. As a last example, a facility consisting of 50 procedures with a total of nearly 4000 PL/1 statements permitting execution of Honeywell 635 programs under Multics became operational after one person spent eight months learning about the GCOS operating system for the 635, PL/1, and Multics, and then implemented the environment. In each example, the implementation was accomplished from remote terminals using PL/1.

Multics users have discovered that it is possible to get their programs running very quickly in this environment. They frequently prepare "rough drafts" of programs, execute them, and then improve their over-all design and operating strategy using the results of experience obtained during actual operation. As an example, again drawn from the implementation of Multics, the early designs and implementations of the programs supporting the virtual memory[18] made overoptimistic use of variable-sized storage allocation techniques. The result was a functionally correct but inadequately performing set of programs. Nevertheless, these modules were used as the foundation for subsequent work for many months. When they were finally replaced with modules using simplified fixed-size storage techniques, performance improvements of more than an order of magnitude were realized. This technique emphasizes two points: first, it is frequently possible to provide a practical, usable facility containing temporary versions of programs; second, often the insight required to significantly improve the behavior of a program comes only after it is

studied in operation. As implied in the earlier discussion of design iteration, our experience has been that structural and strategic changes rather than "polishing" (or recoding in assembly language) produce the most significant performance improvements.

In general, we have noticed a significant "amplifier" or "leverage" effect with the use of an effective online environment as a system programming facility. Major implementation projects on the Multics system seldom involve more than a few programmers, thereby easing the management and communications problems usually entailed by complex system implementations. As would be expected, the amplification effect is most apparent with the best project personnel.

Administration of Multics Facilities and Resources

The problem of managing the capabilities of a computer utility with geographically dispersed subscribers leads to a requirement of decentralized administration. At the apex of an administrative pyramid resides a system administrator with the ability to register new users, confer resource quotas, and generate periodic bills for services rendered. The system administrator deals with user groups called projects. Each group can in turn designate a project administrator who is delegated the authority to manage a budget of system resources on behalf of the project. The project administrator is then free to deal directly with project members without further intervention from the system administrator, thereby greatly reducing the bottlenecks inherent in a completely centralized administrative structure.

Environment Shaping

In addition to having immediate control of such resources as secondary storage, port access, and rate of processor usage, the project administrator is also able to define or shape the environment seen by the members of his project when they log into the system. He does this by defining those procedures that can be accessed by members of his project and by specifying the initial procedure executed by each member of his project when he logs in. This environment-shaping facility has led to the notion of a private project subsystem on Multics. It combines the administrative and programming facilities of Multics so that a project administrator and a few project implementers can build, maintain, and evolve environments entirely on their own. Thus, some subsystems bear no internal resemblance to the standard Multics procedure environment.

For example, the Dartmouth BASIC[19] compiler executes in a closed subsystem implemented by an MIT student group for use by undergraduates. The compiler, its object code, and all support routines execute in a simulation of the native environment provided

at Dartmouth. The users of this subsystem need little, if any, knowledge of Multics and are able to behave as if logged into the Dartmouth system proper. Other examples of controlled environment subsystems include one to permit many programs which normally run under the GCOS operating system to also run unmodified in Multics. Finally, an APL[20] subsystem allows the user to behave for the most part as if he were logged into an APL machine. The significance of these subsystems is that their implementers did not need to interact with the system administrator or to modify already existing Multics capabilities. The administrative facilities permit each such subsystem to be offered by its supporters as a private service with its own group of users, each effectively having its own private computer system.

Other Multics Users

Finally, we observe that the roles of the application user, the system operators, and the hardware maintainers as seen by the system are simply those of ordinary Multics users with specialized access to the online procedures and data. The effect of this uniformity of treatment is to reduce greatly the maintenance burden of the system control software. One example, of great practical importance, has been the ease with which system performance measurement tools have been prepared for use by the operating staff.

INSIGHTS

So far, we have discussed the status and appearance of the Multics system. A further question is what has been learned in the construction of Multics which is useful to other systems designers. Having a bright idea which clearly solves a problem is not sufficient cause to claim a contribution if the idea is to be part of a complex system. In order to establish the real feasibility of an idea, all of its implications and consequences must be followed out. Much of the work on Multics since 1965 has involved following out implications and consequences of the many ideas then proposed for the prototype computer utility. That following out is an essential part of proof of ideas is attested by the difficulties which have been encountered in other engineering efforts such as the development of nuclear fusion power plants and the electric automobile. Not all proposals work out; for example, extended attempts to engineer an atomic-powered airplane suggest infeasibility.

Perhaps Multics' most significant single contribution to the state of the art of computer system construction is the demonstration of a large set of fully implemented ideas in a working system. Further, most of these ideas have been integrated without straining the overall design; most additional proposals would not topple

the structure. Ideas such as virtual memory access to online storage, parallel process organization, routine but controlled information sharing, dynamic linking of procedures, and high-level language implementation have proved remarkably compatible and complementary.

To illustrate some of the areas of progress in understanding of system organization and construction which have been achieved in Multics, we consider here the following five topics:

1. Modular division of responsibility
2. Dynamic reconfiguration
3. Automatically managed multilevel memory
4. Protection of programs and data
5. System programming language

Modular Division of Responsibility

Early in the design of Multics, a decision had to be made whether or not to treat the segmented virtual memory as a separately usable "feature," independent of a traditionally organized read/write type file system. The alternative, to use the segmented virtual memory as the file system itself, providing the illusion of direct "in-core" access to all online storage, was certainly the less conservative approach. (See Fig. 13-2.) The second approach, which was the one chosen, led to a strong test of the ability of a computing system to support an apparent one-level memory for an arbitrarily large information base. It is interesting that the resulting almost total decoupling between physical storage allocation and data movement on the one hand and directory structure, naming, and file organization on the other led to a remarkably simple and functionally modular structure for that part of the system.[18] (See Fig. 13-3.)

Another high degree of functional modularity was achieved in scheduling, multiprogramming, and processor management. Because harnessing of multiple processors was an objective from the beginning, a careful and methodical approach to multiplexing processors, handling interrupts, and providing interprocess synchronizing primitives was developed. The resulting design, known as the Multics traffic controller, absorbed into a single, simple module a set of responsibilities often diffused among a scheduling algorithm, the input/output controlling system, the online file management system, and special purpose interuser communication mechanism mechanisms.[21]

Finally, with processor management and online storage management uncoupled into well-isolated modules, the Multics

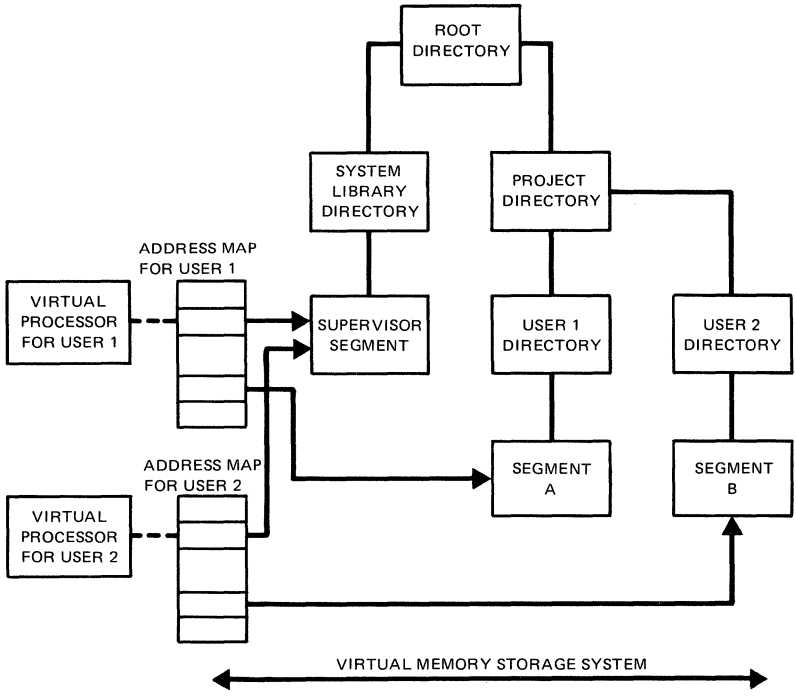


Fig. 13-2. Virtual File System Approach

input/output system was left with the similarly isolatable function of managing streams of data flowing from and to source and sink type devices.[16] Thus, this section of the system concentrates only on switching of the streams, allocation of data buffering areas, and device control strategies.

Each division of labor described above represents an interesting result primarily because it is so difficult to discover appropriate divisions of complex systems.* Establishing that a certain proposed division results in simplicity, creates an uncluttered interface, and does not interfere with performance, is generally cause for a minor celebration.

Dynamic Reconfiguration

If the computer utility is ever to become as much a reality as the electric power utility or the telephone communication service, its continued operation must not be dependent upon any single physical component, since individual components will eventually require maintenance. This observation leads an electric power utility to provide procedures whereby an idle generator may be dynamically added to the utility's generating capacity,

*See Dijkstra[22] for a further discussion of this point.

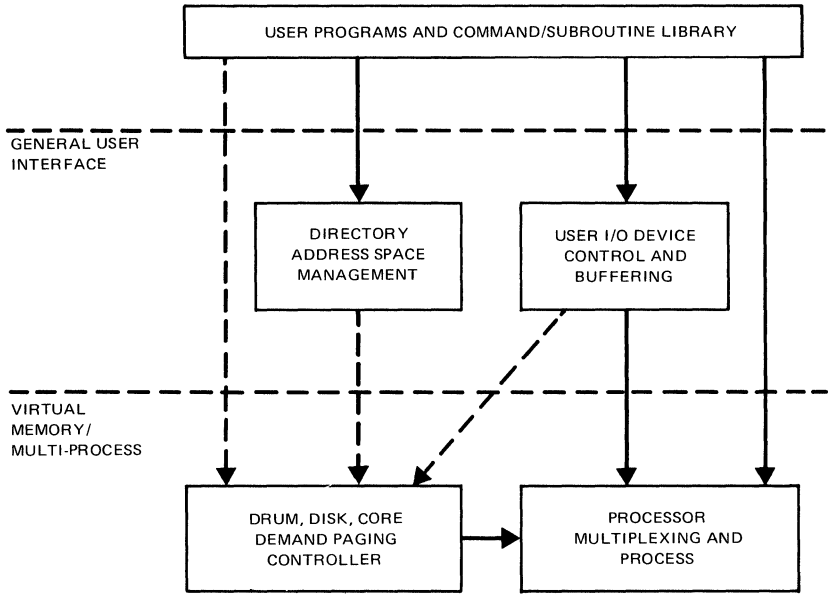


Fig. 13-3. Virtual Memory Feature Approach

while another is removed for maintenance, all without any disruption of service to customers. A similar scenario has long been proposed for multiprocessor, multimemory computer systems, in which one would dynamically switch processors and memory boxes in and out of the operating configuration as needed. Unfortunately, though there have been demonstrated a few "special purpose" designs,* it has not been apparent how to provide for such operations in a general purpose system. A recent thesis[24] proposed a general model for the dynamic binding and unbinding of computation and memory structures to and from ongoing computations. Using this model as a basis, the thesis also proposed a specific implementation for a typical multiprocessor, multimemory computing system. One of the results of this work was the addition to the operating Multics system of the capability of dynamically adding and removing central processors and memory modules as in Fig. 13-4. The usefulness of the idea may be gauged by observing that at MIT five to ten such reconfigurations are performed in a typical

*An outstanding example is the American Airlines SABRE system.[23]

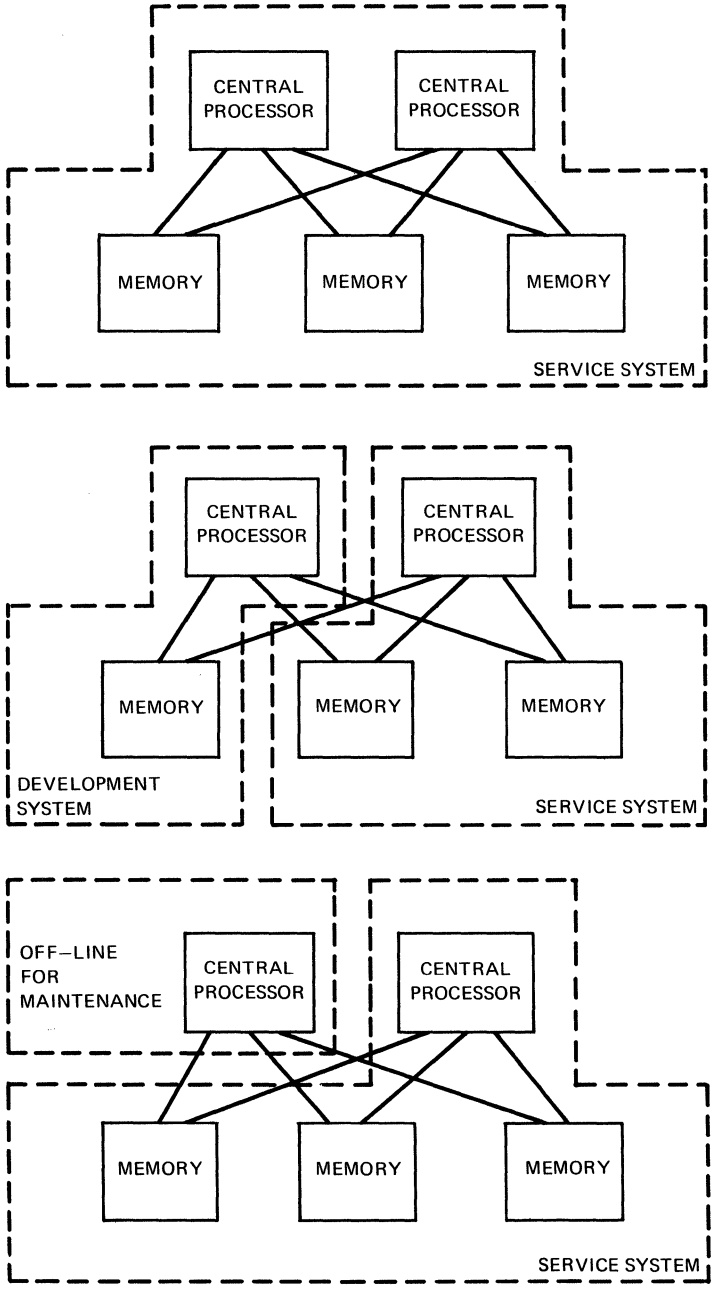


Fig. 13-4. MULTICS Operating System

24-hour operating day. Most of the reconfigurations are used to provide a secondary system for Multics development.

Automatically Managed Multilevel Memory

By now it has become accepted lore in computer systems that the use of automatic management algorithms for memory systems constructed of several levels with different access times can provide a significant reduction of user programming effort. Examples of such automatic management strategies include the buffer memories of the IBM system 370 models 155, 165, and 195[25] and the demand paging virtual memories of Multics, IBM's CP-67[26] and the Michigan Terminal System.[17] Unfortunately, behind the mask of acceptance hides a worrisome lack of knowledge about how to engineer a multilevel memory system with appropriate strategy algorithms which are matched to the load and hardware characteristics. One of the goals of the Multics project has been to instrument and experiment with the multilevel memory system of Multics, in order to learn better how to predict in advance the performance of proposed new automatically managed multilevel memory systems. Several specific aspects of this goal have been explored:

A strategy to treat core memory, drum, and disk as a three-level system has been proposed, including a "least-recently-used" algorithm for moving information from drum to disk. Such an algorithm has been used for some time to determine which pages should be removed from core memory.[27] The dynamics of interaction among two such algorithms operating at different levels are weakly understood, and some experimental work should provide much insight. The proposed strategy will be implemented, and then compared with the simpler present strategy which never moves things from drum to disk, but instead makes educated "guesses" as to which device is most appropriate for the permanent residence of a given page. If the automatic algorithm is at least as good as the older, static one, it would represent an improvement in overall design by itself, since it would automatically track changes in user behavior, while the static algorithm requires attention to the validity of its guesses.

A scheme to permit experimentation with predictive paging algorithms was devised. The scheme provides for each process a list of pages to be preloaded whenever the process is run and a second list to be immediately purged whenever the process stops. The updating of these lists is controlled by a decision table exercised every time the process stops running. Since every page of the Multics virtual memory is potentially shared, the decision table represents a set of heuristics designed to separate out those which are probably not being shared at the moment.

A series of measurements was made to establish the effectiveness of a small hardware associative memory used to hold recently accessed page descriptors. These measurements established a profile of hit ratio (probability of finding a page descriptor in the associative memory) versus associative memory size which should be useful to the designers of virtual memory systems. [28]

A set of models, both analytic and simulation, was constructed to try to understand program behavior in a virtual memory. So far, two results have been obtained. One is the finding that a single program characteristic (the mean execution time before encountering a "missing" page in the virtual memory as a function of memory size) suffices to provide a quite accurate prediction of paging and idle overheads. The second is direct calculation of the distribution of response times under multiprogramming. Having available the entire response time distribution, rather than just averages, permits estimation of the variance and 90-percentile points of the distribution, which may be more meaningful than just the average. A doctoral thesis is in progress on this topic.

Although the immediate effect of each of these investigations is to improve the understanding or performance of the current version of Multics, the long-range payoff in methodical engineering using better understood memory structures is also evident.

Protection of Programs and Data

A long-standing objective of the public computer utility has been to provide facilities for the protection of executing programs from one another, so that users may with confidence place appropriate control on the release of their private information. In 1967, a mechanism was proposed[29] and implemented in software which generalized the usual supervisor-user protection relationship. This mechanism, named "rings of protection," provides user-written subsystems with the same protection from other users that the supervisor has, yet does not require that the user-written subsystem be incorporated into the supervisor. Recently, this approach was brought under intense review, with two results:

A hardware architecture which implements the mechanism was proposed. [30] A chief feature of the proposed architecture is that subroutine calls from one protection ring to another use exactly the same mechanisms as do subroutine calls among procedures within a protection area. The proposal appears sufficiently promising that it is included in

the specifications for the next generation of hardware to be used for Multics.

As an experiment in the feasibility of a multilayered supervisor, several supervisor procedures which required protection, but not all supervisor privileges, were moved into a ring of protection intermediate between the users and the main supervisor. The success of this experiment established that such layering is a practical way to reduce the quantity of supervisor code which must be given all privileges.

Both of these results are viewed as steps toward first, a more complete exploitation and understanding of rings of protection, and later, a less constrained organization of the type suggested by Evans and LeClerc[31] and by Lampson[32]. But more important, rings of protection appear applicable to any computer system using a segmented virtual memory. Two doctoral theses are underway in this area.

System Programming Language

Another technique of system engineering methodology being explored within the Multics project is that of higher level programming language for system implementation. The initial step in this direction (which proved to be a very big step) was the choice of the PL/1 language for the implementation of Multics. By now, Multics offers an extensive case study in the viability of this strategy. Not only has the cost of using a higher level language been acceptable, but increased maintainability of the software has permitted more rapid evolution of the system in response to development ideas as well as user needs. Three specific aspects of this experience have now been completed:

The transition from an early PL/1 subset compiler[14] to a newer compiler which handles almost the entire language was completed. This transition was carried out with performance improvement in practically every module converted in spite of the larger language involved. The significance of the transition is the demonstration that it is not necessary to narrow one's sights to a "simple" subset language for system programming. If the language is thoroughly understood, even a language as complex as the full PL/1 can be effectively used. As a result, the same language and compiler provided for users can also be used for system implementation, thereby minimizing maintenance, confusion, and specialization.

Notwithstanding the observation just made, the time required to implement a full PL/1 compiler is still too great for many

situations in which the compiler implementation cannot be started far enough in advance of system coding. For this reason, considerable interest exists in defining a smaller language which is easily compilable, yet retains the features most important for system implementation. On the basis of the experience of programming Multics in a subset of PL/1, such a language was defined but not implemented, since it was not needed.[33]

A census of Multics system modules reveals how much of the system was actually coded in PL/1, and reasons for use of other languages. Roughly, of the 1500 system modules, about 250 were written in machine language. Most of the machine language modules represent data bases or small subroutines which execute a single privileged instruction. (No attempt was made to provide either a data base compiler or PL/I built-in functions for specialized hardware needs.) Significantly, only a half dozen areas (primarily in the traffic controller, the central page fault path, and interrupt handlers) which were originally written in PL/1 have been recoded in machine language for reasons of squeezing out the utmost in performance. Several programs, originally in machine language, have been recoded in PL/1 to increase their maintainability.

The implications of this work with PL/1 also should be felt far beyond the Multics system. Most implementers, when faced with the economic uncertainties of a higher-level language, have chosen machine language for their central operating systems. The experience of PL/1 in Multics when added to the expanding collection of experience elsewhere[34] should help reduce the uncertainty.

In a research project as large, long, and complex as Multics, any paper such as this must necessarily omit many equally significant ideas and touch only a few which may happen to have wide current interest. The purpose of individual and detailed technical papers is to explain these and other ideas more fully. The bibliography found in reference[35] contains more than 20 such technical papers.

Immediate Future Plans

Multics software is continuing to evolve in response to user needs and improved understanding of its organization. In 1972, a new hardware base for Multics will be installed by the Information Processing Center at MIT for use by the MIT computing community. This program compatible hardware base contains small but significant architectural extensions to the current hardware. The circuit technology used will be that of the Honeywell 6080 computer. The

substantial changes include: (1) replacement of the high-performance paging drum initially with bulk core and, when available, LSI memory, and (2) implementation of rings of protection as part of the paging and segmentation hardware.

Wherever possible the strategy of using off-the-shelf standard equipment rather than specially engineered units for Multics has been followed. This strategy is intended to simplify maintenance.

CONCLUSIONS

Many conclusions could possibly be drawn from the experience of the Multics project. Of these, we consider four to be major and worthy of note. First, we feel it is clear that it is possible to achieve the goals of a prototype computer utility. The current implementation of Multics provides a measure of the mechanisms required. Moreover, the specific implementation of the system, because it has been written in PL/1, forms a model for other system designers to draw upon when constructing similar systems.

Second, the question of whether or not the specific software features and mechanisms which were postulated for effective computer utility operation are desirable has now been tested with specific user experience. Although the specific mechanisms implemented subsequently may be superseded by better ones, it is certainly clear that the improvement of the user environment which was wanted has been achieved.

Third, systems of the computer utility class must evolve indefinitely since the cost of starting over is usually prohibitive and the many-year lead time required may be equally unacceptable. The requirement of evolvability places stringent demands on design, maintainability, and implementation techniques.

Fourth and finally, the very act of creating a system which solves many of the problems posed in 1965 has opened up many new directions of research and development. It would appear almost a certainty that increased user aspirations will continue to require intensive work in computer system principles and techniques.

In closing, perhaps we should take note that in the seven years since Multics was proposed, a great many other systems have also been proposed and constructed; many of these have developed similar ideas.* In most cases, their designers have

---

*Some examples which have not already been mentioned include: the TENEX system of Bolt, Beranek and Newman; the VENUS system of Mitre Corp.; the MU5 at Manchester University; RC-4000 of Regnecentralen; 5020 TSS of Hitachi Corp.; DIPS-1 of Nippon Telephone; the Japanese National Computer Project; the

developed effective implementations which are directed to a different interpretation of the goals, or to a smaller set of goals than those required for the complete computer utility. This diversity is valuable, and probably necessary, to accomplish a thorough exploration of many individually complex ideas, and thereby to meet a future which holds increasing demand for systems which embrace the totality of computer utility requirements.

ACKNOWLEDGEMENT

It is impossible to acknowledge accurately the contributions of all the individuals or even the several organizations which have given various forms of support to Multics development over the past seven years. As would be expected of any multiorganization project spanning several years, there has been a turnover in personnel. As the individual contributors now number in the hundreds, proper recognition cannot be given here. Instead, since the development of significant features and designs of Multics has occurred in specific areas and disciplines such as input/output, virtual memory design, languages, and resource multiplexing, a more accurate delineation of achievements should be made in specialized papers. So in the end we must defer to the authors of individual papers, past and future, to acknowledge the efforts of some of the many contributors who have made possible Multics evolution.

PDP-10/50 TSS of Digital Equipment Corp.; the BCC-500 of Berkeley Computer Corp.; I. T. S. of MIT Artificial Intelligence Laboratory; Exec-8 of Univac; System 3 and 7 and the SPECTRA 70/46 of RCA; Star-100 of CDC; UTS of Xerox Data Systems; the 6700 system of Burroughs, and the Dartmouth Time-Sharing System.

REFERENCES

1. Corbató, F. J., Daggett, M. M., and Daley, R. C., "An Experimental Time-Sharing System", AFIPS Conf. Proc. 21, Spartan Books, 1962, pp. 335-344.

2. Crisman, P. A., ed., "The Compatible Time-Sharing System: A Programmer's Guide", 2nd ed., M.I.T. Press, Cambridge, Massachusetts, 1965.

3. Corbató, F. J., and Vyssotsky, V. A., "Introduction and Overview of the Multics System", AFIPS Conf. Proc. 27 (1965 FJCC), Spartan Books, Washington, D. C., 1965, pp. 185-196.

4. Glaser, E. L., et al., "System Design of a Computer for Time-Sharing Application", AFIPS Conf. Proc. 27 (1965 FJCC), Spartan Books, Washington, D. C., 1965, pp. 197-202.

5. Vyssotsky, V. A., et al., "Structure of the Multics Supervisor", AFIPS Conf. Proc. 27 (1965 FJCC), Spartan Books, Washington, D. C., 1965, pp. 203-212.

6. Daley, R. C., and Neumann, P. G., "A General-Purpose File System for Secondary Storage", AFIPS Conf. Proc. 27 (1965 FJCC), Spartan Books, Washington, D. C., 1965, pp. 213-229.

7. Ossanna, J. F., et al., "Communication and Input/Output Switching in a Multiplex Computing System", AFIPS Conf. Proc. 27 (1965 FJCC), Spartan Books, Washington, D. C., 1965, pp. 231-241.

8. David, E. E., Jr., and Fano, R. M., "Some Thoughts About the Social Implications of Accessible Computing", AFIPS Conf. Proc. 27 (1965 FJCC), Spartan Books, Washington, D. C., 1965, pp. 243-247.

9. Organick, E. I., The Multics System: An Examination of its Structure, M.I.T. Press (in press), Cambridge, Massachusetts and London, England.

10. Watson, R. W., Timesharing System Design Concepts, McGraw-Hill Book Company, New York, 1970.

11. Comfort, Webb T., "A Computing System Design for User Service", AFIPS Conf. Proc. 27 (1965 FJCC), Spartan Books, Washington, D. C., 1965, pp. 619-626.

12. Lett, A. S., and Konigsford, W. L., "TSS/360: A Time-Shared Operating System", AFIPS Conf. Proc. 33 (1968 FJCC), Thompson Books, pp. 15-28.

13. Schwemm, R. E., "Experience Gained in the Development and Use of TSS/360", AFIPS Conf. Proc. 40 (1972 SJCC), AFIPS Press. (This volume.)

14. Corbató, F. J., "PL/1 as a Tool for System Programming", Datamation 15, 6 (May, 1969) pp. 68-76.

15. Freiburghouse, R. A., "The Multics PL/1 Compiler", AFIPS Conf. Proc. 35 (1969 FJCC), AFIPS Press, 1969, pp. 187-199.

16. Feiertag, R. J., and Organick, E. I., "The Multics Input-Output System", ACM Third Symposium on Operating Systems Principles, (October 18-20, 1971), pp. 35-41.

17. Alexander, M. T., "Organization and Features of the Michigan Terminal System", AFIPS Conf. Proc. 40 (1972 SJCC), AFIPS Press. (This volume.)

18. Bensoussan, A., Clingen, C. T., and Daley, R. C., "The Multics Virtual Memory", ACM Second Symposium on Operating System Principles, (October 20-22, 1969) Princeton University, pp. 30-42.

19. BASIC, Fifth Edition, Kiewit Computation Center, Dartmouth College (September, 1970).

20. APL/360 User's Manual, IBM form number GH20-0683-1 (March, 1970).

21. Saltzer, J. H., "Traffic Control in a Multiplexed Computer System", ScD. Thesis, M. I. T. Department of Electrical Engineering, 1966. Also available as Project MAC technical report TR-30.

22. Dijkstra, E. W., "The Structure of the 'THE'-Multiprogramming System", Comm. ACM 11, 5 (May, 1968), pp. 341-346.

23. Parker, R. W., "The Sabre System", Datamation 11, 9, September, 1965, pp. 49-52.

24. Schell, R. R., "Dynamic Reconfiguration in a Modular Computer System", Ph. D. Thesis, M. I. T. Department of Electrical Engineering, 1971. Also available as Project MAC technical report TR-86.

25. Conti, C. J., "Concepts for Buffer Storage", IEEE Computer Group News, March, 1969, pp. 9-13.

26. Meyer, R. A., and Seawright, L. H., "A Virtual Machine Time-Sharing System", IBM Systems Journal 9, 3, 1970, pp. 199-218.

27. Corbató, F. J., "A Paging Experiment with the Multics System", In Honor of P. M. Morse, M.I.T. Press, Cambridge, Massachusetts, 1969, pp. 217-228.

28. Schroeder, M. D., "Performance of the GE-645 Associative Memory While Multics is in Operation", ACM Workshop on System Performance Evaluation (April, 1971), pp. 227-245.

29. Graham, R. M., "Protection in an Information Processing Utility", Comm. ACM 11, 5 (May, 1968) pp. 365-369.

30. Schroeder, M. D., and Saltzer, J. H., "A Hardware Architecture for Implementing Protection Rings", ACM Third Symposium on Operating Systems Principles, (October 18-20, 1971), pp. 42-54.

31. Evans, D. C., and LeClerc, J. Y., "Address Mapping and the Control of Access in an Interactive Computer", AFIPS Conf. Proc. 30, (1967 SJCC), Thompson Books, 1967, pp. 23-30.

32. Lampson, Butler W., "An Overview of the CAL Time-Sharing System", Computer Center, University of California, Berkeley (September 5, 1969).

33. Clark, D. D., Graham, R. M., Saltzer, J. H., and Schroeder, M. D., "Classroom Information and Computing Service", M.I.T. Project MAC Technical Report TR-80, (January 11, 1971).

34. Sammet, Jean E., "Brief Survey of Languages Used for Systems Implementation", SIGPLAN Notices 6, 9, October, 1971, pp. 1-19.

35. The Multiplexed Information and Computing Service: Programmers' Manual, M.I.T. Project MAC, Rev. 10, 1972. (Available from the M.I.T. Information Processing Center.)

14. Design of the Venus Operating System

B. H. Liskov

Massachusetts Institute of Technology
Cambridge, Massachusetts

The Venus operating system is an experimental multiprogramming system* designed to support five or six interactive users on a small computer. The system was primarily intended to support users who are cooperating with one another – for example, sharing a data base. It was produced as a product of a project performed at MITRE under the sponsorship of the Electronic Systems Division (ESD) of the Air Force.

The primary hypothesis of this project was that software complexity can be reduced by using a machine with "correct" machine architecture. We had in mind some kind of complex system in which the data and processing requirements vary dynamically; for example, an online data management system or an operating system or any kind of system supporting interactive users. We felt that the architecture of current-day computers does not support the programming of such systems very well and, in fact, that considerable complexity is introduced into the software to compensate for the inadequacies of the hardware.

The first step was to define what we meant by "correct" machine architecture, and then to produce a machine with this architecture. This was done by microprogramming on an Interdata 3 computer. The result was the Venus machine. Then, given the Venus machine, the next step was to produce a software system. An operating system was selected for the experiment.

This chapter concentrates on describing the development of the system according to certain system design principles.

One important system design principle was that the system be designed as a hierarchy of levels of abstraction. Levels of

---

*A fuller description of the system is contained in the paper[1] on which this chapter is based.

abstraction were first introduced by Dijkstra.[2] A level of abstraction is defined by the abstraction it supports (for example, virtual memories). In addition, a level of abstraction is defined by the resources it owns, and there are very strict rules concerning resource access. First, lower levels of abstraction, those that are closer to the machine, are not aware of the existence of higher levels of abstraction and, obviously, are therefore not aware of the resources of higher levels. Higher levels are aware of the existence of lower levels and may need to obtain information contained in the resources of lower levels, but they cannot access those resources directly. Instead, they will have to appeal to the functions of lower levels to obtain desired information. The use of levels of abstraction was applied both to the microprogram and to the software; examples of levels will be given later.

The other important design principle was to allow the features of the machine architecture to have a direct influence on the software design. This principle was selected to enable us to evaluate the basic hypothesis of the project; that is, that correct machine architecture can reduce the complexity of software.

THE VENUS MACHINE

Before describing the software of the Venus operating system, I will briefly describe some features of the Venus machine itself. It was constructed by microprogramming an Interdata 3 computer, which is a very small, slow computer. The basic instruction set which the microprogram supports is a relatively ordinary one for a small computer; however, it was augmented by certain special instructions and also by some special data structures. Two features of the Venus machine are especially useful to the software: multiprogramming and segments.

The Venus microprogram supports the multiprogramming of 16 concurrent processes. A process is defined to be a procedure in execution on a virtual machine. So the microprogram supports 16 virtual machines, each one consisting of an address space and a work area. The address space is primarily made up of segments; the work area contains information about the state of the process; it is permanently located in core memory.

The microprogram performs the scheduling of the CPU to the 16 virtual machines. It does this only as a result of operations being performed on semaphores. Semaphores are special types of data which were first defined by Dijkstra.[2] Only two operations can be performed on semaphores – the P operation and the V operation. The P operation is performed when a process wishes to wait either for a resource to become available or for an event to occur. A V operation is performed for the opposite reason – to release a resource or to signal that an event has occurred. Whenever a P or V operation is performed by some process, the

microprogram will reexamine the scheduling of the CPU to the processes and possibly change the state of some of the processes.

There is a microprogrammed multiplexed I/O channel on the Venus machine, primarily provided to relieve the software of any real-time constraints associated with the I/O devices. The only significant thing about this channel is that it cooperates with the microprogram, as far as the scheduling is concerned, by signaling the end of an I/O transfer by performing a V on a semaphore. There are no I/O interrupts at the software level on this machine.

Segments on the Venus machine are named virtual memories – as are the segments in Multics.[3] Segments have 15-bit names and may contain up to 64 thousand bytes of data. The microprogram performs the mapping of segment addresses into core addresses. Segments are divided into 256-byte pages and so is core memory; segments are paged on demand between core and the disk.

Segments are physically shared among the users of the system, making it very easy for users to share data in segments. On the other hand, it is impossible to protect segments without the cooperation of the users involved. This is accpetable only because the system is experimental and was designed specifically to support cooperating users.

Segments are the primary storage structure available on the Venus machine; they are used to hold both data and procedures. Since procedures are stored in segments, they will be physically shared by the users; and therefore, it is important that they be reentrant. The Venus machine provides support for reentrant procedures through a reentrant procedure interface consisting of call and return instructions which save and restore the state of a process and push-down stacks for holding arguments and values of procedures. In addition, reentrancy is enhanced by not providing an easy way of storing into procedures.

Levels of abstraction supported by the Venus microprogram include:

ABSTRACTION	RESOURCES	METHOD OF APPEAL
Segments	Core, disk	Segment reference, EL1 instruction
Virtual devices	Devices, device status word table	SIO instruction, channel commands
Virtual machines	CPU cycles	P and V instructions

The levels support the segments, the virtual devices, and the virtual machines. The level of abstraction supporting segments has as its resources core memory and the paging disk,

and entry into the level occurs whenever a segment is referenced or an EL1 instruction is executed. The EL1 instruction was defined because of the rule about resources of levels of abstraction. We knew that the tables owned by this level of abstraction contained valuable information which the software would need, and the software would not be able to access that information directly. So we defined the EL1 instruction to permit the software to appeal to the level to obtain that information.

The virtual devices are supported by the microprogrammed channel, and the resources of interest are the real devices, and also the device status word table which tells about the state of each device. Entry into this level happens as a result of an SIO (start I/O) instruction together with a sequence of channel commands describing the transfer to take place.

In virtual machines, the important resource is the CPU cycles being distributed among the processes. Entry into this level happens only as the result of performance of P and V instructions.

The levels shown previously are given in their order in the hierarchy of levels. The lowest level supports virtual machines; the next, the virtual devices; and the highest, the segments. The virtual device level is higher than the virtual machine level because it performs a V on a semaphore to signal the end of an I/O transfer. The segment level is higher than the virtual device level because it calls upon the virtual device level in order to perform the transfer of segment pages between core and disk.

Now, the major features of the machine architecture can be examined from the point of view of building a software system on the computer. One important thing about the machine architecture is that there are 16 processes available. We were thinking of supporting five or six users; in fact, we have only five interactive devices. Assuming each user has his own process, several processes are still left over for system use. Now, in an operating system there are many tasks being performed which are logically asynchronous with each other and also with the users. For example, a part of the system manages the I/O devices for the system as a whole, and this is logically asynchronous with the user, who is concerned only with his own devices. Having several processes at our disposal meant that logically asynchronous tasks could be made to perform in a physically asynchronous manner by assigning them to separate processes. This led to clarity and reduced complexity in the design.

Another important feature of the Venus machine is that procedures and data can be shared. Therefore, before beginning to design the system, we expected it to be composed of reentrant procedures running as independent processes and using segments to hold shared and private data.

An important part of any operating system is the resource management. Resource management is necessary on a multiuser system to avoid the chaos which would result if the users were to compete freely for the resources. Goals of resource management are to treat the users fairly and to use the resources efficiently for the system as a whole.

Also, in our system we were anxious to prevent deadlocks from occurring on the resources controlled by the system. As an example of deadlock, suppose there are two processes: Process One, which owns the tape and must have the printer in order to continue, and Process Two, which owns the printer and must have the tape in order to continue. Process One and Process Two are in a deadlock in which neither can continue; in addition, the system suffers because neither the tape nor the printer is available. One way to resolve this deadlock is to remove either Process One or Process Two. This is not always a good idea, however, particularly in a system such as ours in which users may share data. For example, one of the processes involved in a deadlock may have been manipulating some common data in such a way that the data is inconsistent and only that process can make it consistent again. Removing that process to resolve a deadlock is obviously unsatisfactory for the other users of the data.

Core, the disk, and the CPU cycles are system resources which are managed by the microprogram; and therefore, we did not need to worry about them in software. We still had to manage the I/O devices, however, and also the segments. Segments are used to hold data and procedures. Procedures are all reentrant; and therefore, no management was necessary. A reentrant procedure can be run from all 16 processes at once without any difficulties arising. We still had to be concerned with the management of data segments.

We distinguished between system-defined data segments and user-defined data segments. An example of system-defined data segments is provided by the dictionaries, which contain mappings between external symbolic names for segments and the internal segment names recognized by the microprogram. The dictionaries are intended to be shared because we expect the users to share segments. Furthermore, when a dictionary is being changed, it does not contain consistent data. For example, each dictionary contains a count of the number of entries. While an entry is being added or deleted, there will be a time at which the count does not agree with the actual number of entries. In other words, a need exists here for mutual exclusion. Mutual exclusion can be provided by attaching a semaphore to each dictionary and then performing a P on the semaphore before starting to use the dictionary and a V when finished. However, if the P's and V's are not performed correctly – for example, if a P is performed and the corresponding V is not – then the dictionary will become unavailable to the system as a whole.

Our solution to the management of such system data segments is to limit access to only a special group of functions, which perform P's and V's correctly (the system-defined dictionary functions in the case of dictionaries). But this is the same as treating the segments like the resource of a level of abstraction; naturally, access to the resource will be limited to the functions which make up the level. There are several other groups of system data segments like the dictionaries, and these segments are all managed in this way.

In general, the management of shared user-defined data segments is a very difficult problem. For one thing, there are many different types of sharing, and it is difficult to say which type is best in all circumstances. We decided not to try to control access to these segments. Instead, because the P's and V's exist, the users have at their disposal a tool which will allow them to define an algorithm of their own choosing if they so wish.

The system's management of teletypes will serve as an example of management of I/O devices and also illustrate how the work of the system is performed by reentrant procedures distributed over independent processes. In order to manage devices, it is first necessary to solve the problem of how they are going to be used. Our solution involves the notion of a "preferred" teletype. This is the teletype at which the user sits down and logs in. His programs can refer to this teletype symbolically. However, he does not own the teletype. He does control it for limited transactions – generally long enough for his program to ask him a question and for him to respond with a command. But between these transactions, he cannot prevent either the system or other users from sending messages to his teletype.

Once the problem of device use is solved, the next problem is somehow to support this use. In our system we support the notion of "preferred" teletype by means of three levels of abstraction, each one of which can be thought of as supporting a virtual device with different characteristics. These levels of abstraction primarily are made up of shared reentrant procedures running as independent processes.

The lowest level of abstraction is the microprogrammed channel. The characteristics of the virtual device at this level are very similar to those of the real teletypes, except that data may be transferred many bytes at a time. Core buffers are required to hold the data being transferred, and the completion of I/O is signalled by the performance of a V on a semaphore which is also located in core.

The first level defined in software is made up of teletype controllers. There is one teletype controller per teletype; each controller runs as an independent process. But there is a single reentrant teletype controller procedure. The characteristics of the virtual device at this level are the following: The types of transfers performed by the "preferred" teletype are defined at

this level; most important is the standard interactive transfer in which a program writes out a line and then the user responds with a command. The I/O buffers holding the data to be transferred are now in segments. A very important characteristic is that the virtual device may be addressed whether it is busy or not; a request for a transfer is put on a queue, and the teletype controller will satisfy the request when the device is available. And finally, the teletype controller is informed of the location of the semaphore on which it will perform a V when the transfer is finished; generally this semaphore will be located in a segment.

One final level completes the notion of "preferred" teletype; this level consists of the teletype requester. Again this is a single reentrant procedure, but now it runs on the virtual machine of the user who is requesting the I/O transfer. The characteristics of the virtual device at this level are very similar to those supported by the teletype controller; however, there is now a simplified interface. The user can describe the transfer to the teletype requester by means of a few arguments at the time of the call, whereas the teletype controller requires a fairly complicated request element describing the transfer. One argument of the teletype requester is the symbolic reference to the "preferred" teletype. In addition, the teletype requester defines some rules about how a user can use the teletypes; in particular, he must finish I/O on a given teletype before starting a new transfer on the same teletype.

An example of how a user or system function performs teletype I/O on our system will illustrate how control passes among the various levels of abstraction. The teletype requester is called with arguments describing the transfer to take place. It builds a request element, puts it on a queue, and then notifies the teletype controller by performing a V on a semaphore. When the teletype is available, the teletype controller picks up the request element, probably moves some data from a segment buffer to a core buffer, and starts the microprogrammed channel by performing an SIO instruction. When the I/O transfer is complete, the microprogrammed channel performs a V on a core semaphore associated with the teletype controller. The teletype controller moves data from a core buffer and then notifies the teletype requester that the I/O is complete by performing a V on the semaphore named in the request element. The teletype requester then returns to the user.

Actually, it is not necessary for the user or system function to wait for the completion of I/O. Different arguments will cause the teletype requester to return as soon as the transfer has been initiated. Then when the user is ready to find out the result of the transfer, he calls the teletype requester again. This time the teletype requester will wait for the completion of the I/O and then return to the user.

CONCLUSIONS

Although it is extremely difficult to evaluate a project of this sort, qualitatively the results seem good; the design of the system is clear and consistent. Quantitatively, it required about two man-years to build the microprogram which supports the Venus machine. Then, given this machine, it took about six man-years to build the software part of the system. This includes not only the operating system functions like the resource management but also a large number of utility functions. For instance, there is an assembler, an online interactive symbolic editor, and a large complement of debugging aids and instrumentation at the software level. Six man-years is a rather brief time to develop a system of this complexity. This result appears to support our basic hypothesis that the complexity of software can be reduced by having a machine with the correct architecture. In addition, it seems that we had a fairly good idea of what the correct architecture should be. Also, the levels of abstraction proved to be very valuable because they gave us a clear and precise way of thinking about the structure of the system.

REFERENCES

1. B. H. Liskov, "The Design of the Venus Operating System," to be published in Communications of the ACM, March 1972.

2. E. W. Dijkstra, "The Structure of the 'THE' – Multiprogramming System," Communications of the ACM, 11, 5, May 1968, 341-346.

3. F. J. Corbató and V. A. Vyssotsky, "Introduction and Overview of the Multics System," Proceedings AFIPS 1965 Fall Joint Computer Conference, 27, Part 1, Spartan Books, New York, 185-196.

15. Interactive Computer-Controlled Information Television (TICCIT)

John Volk

The MITRE Corporation

McLean, Virginia

At MITRE we are now developing a new style of time sharing. We call it Time-shared Interactive Computer Controlled Information Television. It's new, not so much in technological approach but in its intended users. We hope to develop a system that provides help, education, and entertainment for the masses of urban America. We call it TICCIT.

This program began several years ago as an IR&D project to investigate ways in which MITRE's aerospace skills could be applied to the problems of education. One outcome of this work was a CAI system, much lower in cost than any other system and surprisingly powerful. An ambitious program to evaluate two of these systems is now under way under NSF sponsorship. During the design of our CAI system we soon came across the problem of bringing CAI not only to the child in the school but also to the child in the home. The solution to this problem turned out to be the vehicle for an expanded, more revolutionary effort.

The contents of a time-sharing system for all ages and classes of individuals must be as diversified and interesting as the individuals themselves. One major effort we are about to begin will be an assessment of the wide variety of possible services (Fig. 15-1). In about two years, we will actually market-test selected services in a real system.

Our technical approach to time sharing for the masses is quite simple (Fig. 15-2). We build a minicomputer system, use a cable TV to serve as a communication link to each home (Fig. 15-3). A standard home TV set is used as a display. A single channel on the CATV system is time-diversion multiplexed to deliver pictures to multiple users. First, a picture is sent to one user in a 60th of a second, then to another, then to a third and so on. A device to catch pictures is installed at each home. This device captures the picture in a 60th of a second and then repetitively plays it back

PROBLEM SOLVING
- DESK CALCULATION
- SOCIAL SECURITY
- HIGHER LEVEL MATHEMATICS

DAILY ROUTINES
- BANKING
- TELE-SHOPPING

EDUCATION AND INFORMATION
- PROGRAMMED LEARNING
- MUNICIPAL GOV'T REGULATIONS
- CRIME DETECTION – "MUG SHOTS"
- PARTICIPATE IN SCHOOL BD. MEETINGS – CHEER, BOO, QUESTION
- PLATFORM FOR PERSONAL OPINIONS
- GAME-PLAYING FOR SHUT-INS
- SOCIAL SERVICE "YELLOW PAGES"
- PICTUREPHONE "FACE-TO-FACE" COMMUNICATION
- ELECTION-TIME ISSUES AND ANSWERS
- OFFTRACK BETTING

Fig. 15-1. Potential TICCIT Home Services

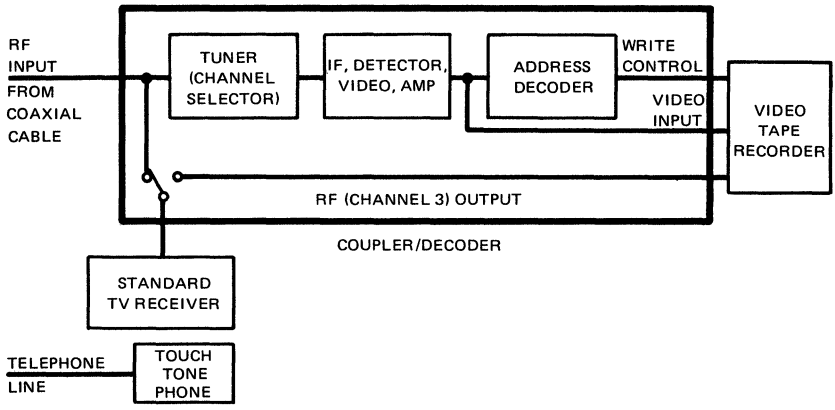


Fig. 15-2. Demonstration System Home Terminal

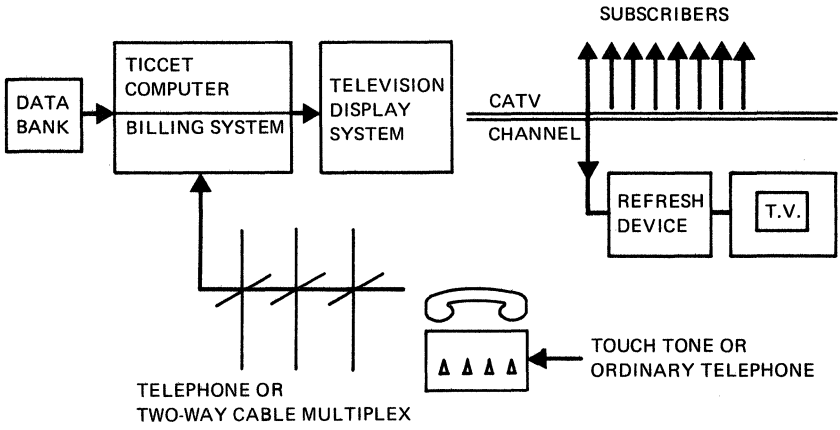


Fig. 15-3. TICCET CATV Application

so that it appears to home TV sets as a normal television signal. A keyboard is also provided at each home to allow the user to communicate with the computer. We have built such a system and are demonstrating it almost daily in Reston, Virginia (Fig. 15-4).

In each demonstration home, we have installed a video tape recorder and what we call a coupler/decoder (Fig. 15-5). The video tape recorder, located on the top of the TV set on the left side, operates in a stop frame mode. In other words, when the reels start moving, the tape is stationary but the head of the video tape recorder still passes over the tape. This device in this mode serves as picture catcher. A touch-tone phone is used as a link back to the computer.

The coupler/decoder (Fig. 15-6) examines TV pictures sent on the TICCIT channel and causes the video tape recorder to capture selected pictures. The demonstration system has both noninteractive and interactive services. The noninteractive, or public, pictures are sent cyclically about 200 to 300 different pictures every five seconds. These pictures include information that has both topicality and wide interest, such as weather reports, stock market news, game reports, etc. To view these, the public-private switch is placed in the public position and the subchannel switch is set to the code of the desired information. In the interactive mode, the switch is set in the private position.

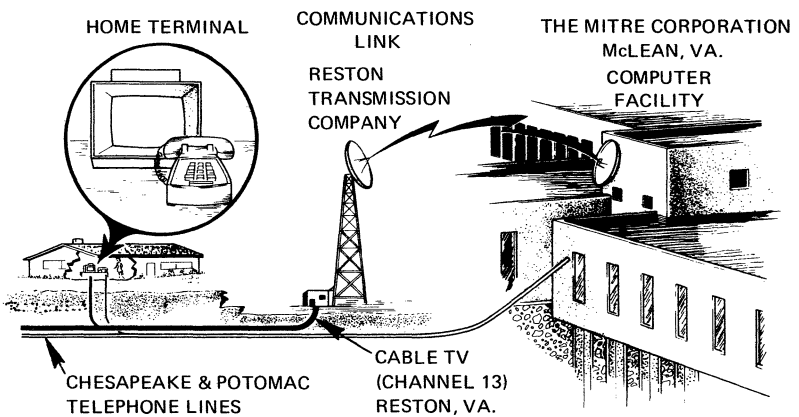


Fig. 15-4. Reston Test Overall System Block Diagram.

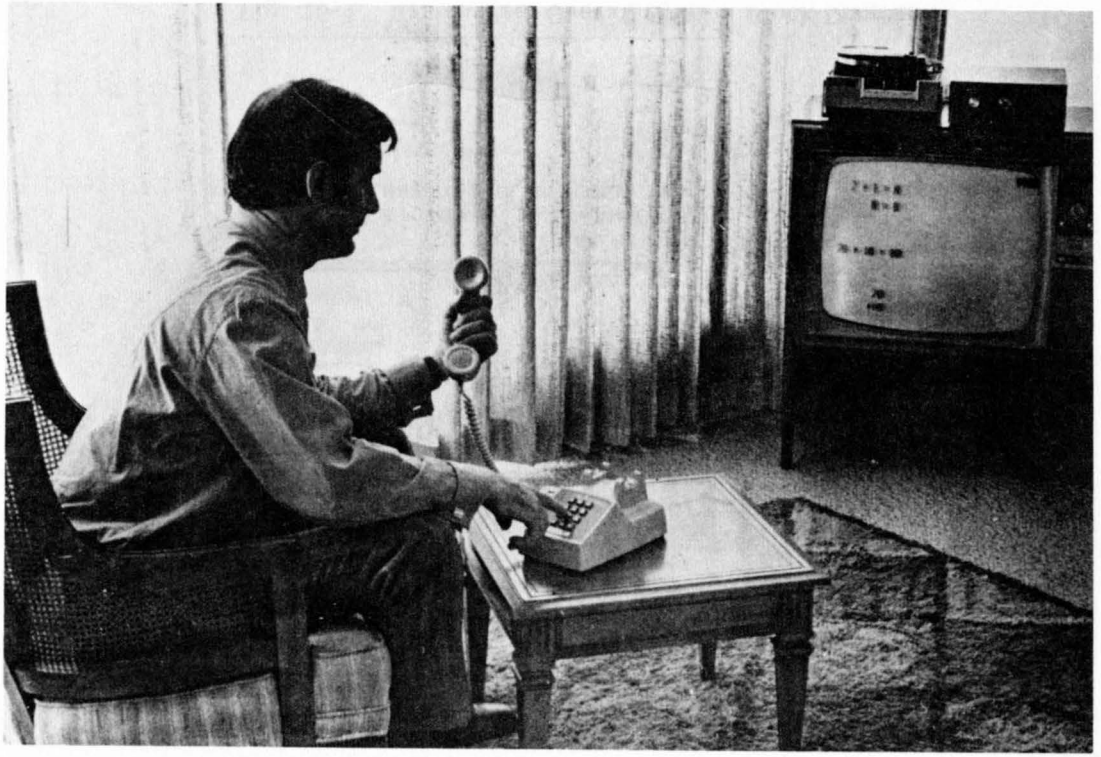


Fig. 15-5. Interactive Home Terminal.

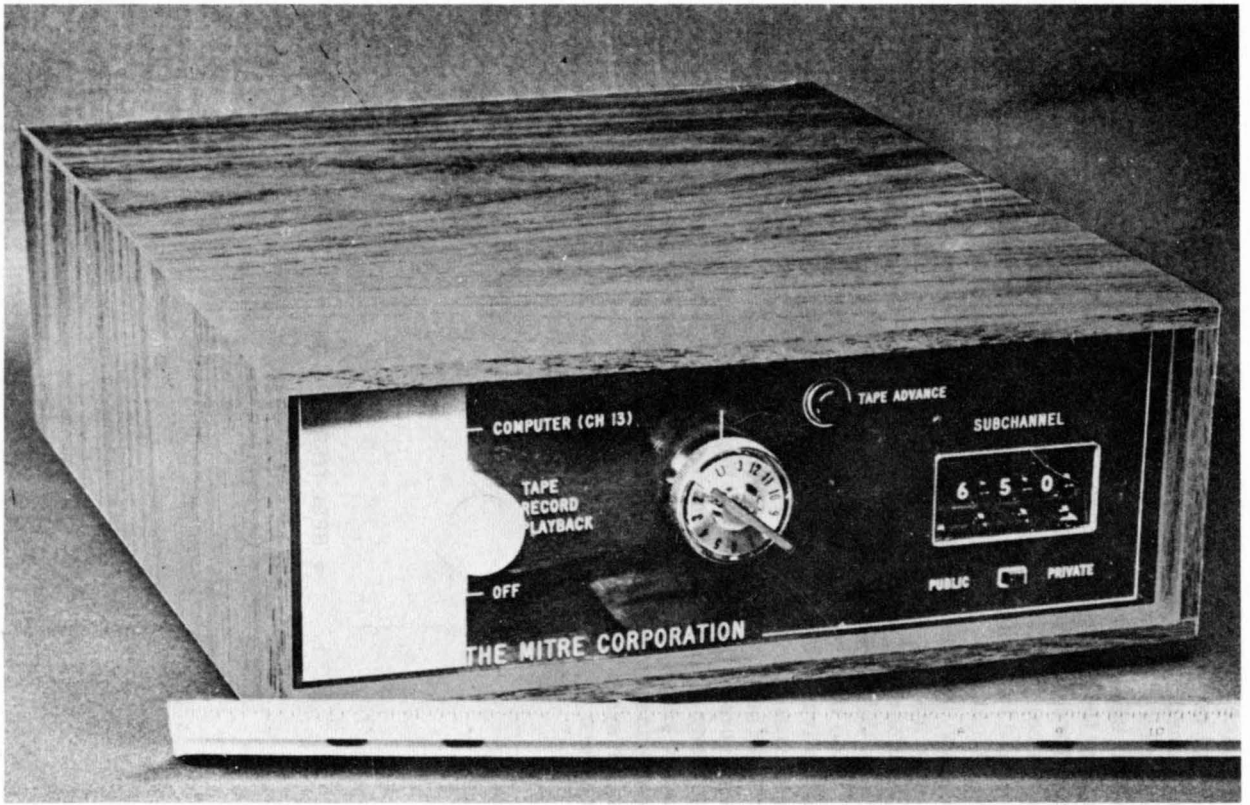


Fig. 15-6. Mitre Coupler/Decoder

This position captures only pictures generated specifically for the home.

Each picture assembled by the computer contains an address (a 16-bit code of white dashes on a black background on a scan line immediately preceding the vertical retrace interval of a standard TV picture). In the public mode, the coupler/decoder looks at each address received and determines if that address matches the subchannel code. In the private mode, it examines the pictures to determine if the specific home address was set. When a correct address is detected, the tape recorder is forced to record for a 60th of a second. It thereby captures the following picture and then switches automatically back to playback and continuously displays the captured picture on the home TV.

In addition to using video tape recorders for the home refresh device, we are also investigating the use of other devices such as the image storage tubes, cassette video tape recorders and video disc recorders (Fig. 15-7). Using video tape recorders was motivated by the possibility that many consumers will purchase cassette video tape recorders in the next few years, purely for their entertainment value. The cost of interactive TV then would be only the cost of the inexpensive coupler/decoder. The cost today of a video tape recorder though is approximately $1000. In a few years, the cost should drop to the $400 to $700 range. The Japanese are also interested in this and are developing a similar system. They are proposing to use a video disc recorder to capture pictures. Their estimated cost for this device is about

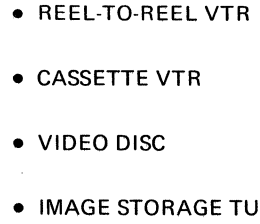


Fig. 15-7. Video Refresh Devices.

$350. RCA is also interested. They propose to use electronic storage tubes as frame catchers at a cost of about $100. In addition, we are considering near-term alternatives such as all-digital character memory and character generator located in each home rather than a full-scale frame catcher with pictorial capability. It appears that a device of this type with only alpha-numeric capabilities could be mass produced today for approximately $20.

In our demonstration system, the telephone is used as the communication link back to the computer (Fig. 15-8). We intend to use the same coaxial cable that carries the TV picture to the home to also carry keyboard signals back to the computer. One way we are considering to facilitate this in the future is to frequency division multiplex the cable. High frequencies (TV channel 2 and up) are amplified in the forward direction, that is, from the computer to the home. Low frequencies, below 50 megacycles, are amplified only in the return direction. Cable television equipment manufacturers are today manufacturing the required amplifier and filters to implement this technique and, in fact, CATV systems are being designed and built to provide this capability.

We are now beginning to address the problem of providing the necessary computer power to serve large groups of people. (Systems serving 30,000 homes have been proposed.) One approach is a distributive computer network with a hierarchy of data bases (Fig. 15-9). Remote computers would serve 1000 to 2000 homes, with perhaps 10 percent of the terminals active at any one time. The central computer would serve clusters of ten or so remote computers providing supervisory control and as a central data entry point. The remote processors would have data bases containing commonly requested data. Specialized data such as CAI course modules would be stored at the central computer.

The scope of the central computer is not yet clear. However, we do have a feel for the remote computer configuration (Fig. 15-10). The configuration would cost approximately $200,000 to $300,000 today and possibly could drop to $150,000 by 1975. Minicomputers would be used for both the main and the terminal processors (2314 type disk files for the main data base). The virtual memory isn't really a virtual memory but a pair of fixed-head disk memories with a special program. The character generator, the keyboard signal processor, and the audio message generator are devices that MITRE is designing and developing today.

Several points should be made in closing. One is that while the system may seem technically naive, those who come out to see the system are just overwhelmed. Those in higher management positions and in high governmental positions come out and sit in the home where we demonstrate the system and occasionally go away almost in shock. They had not realized that anything like this could even remotely happen in the near future.

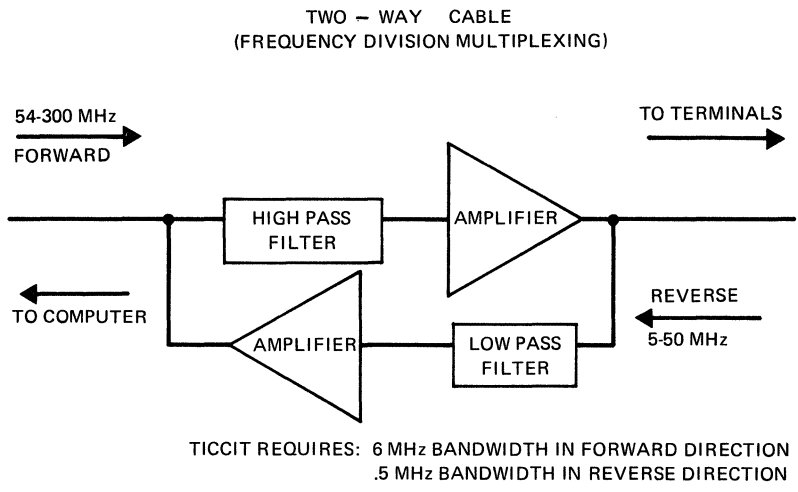


Fig. 15-8. TICCET Communications Link

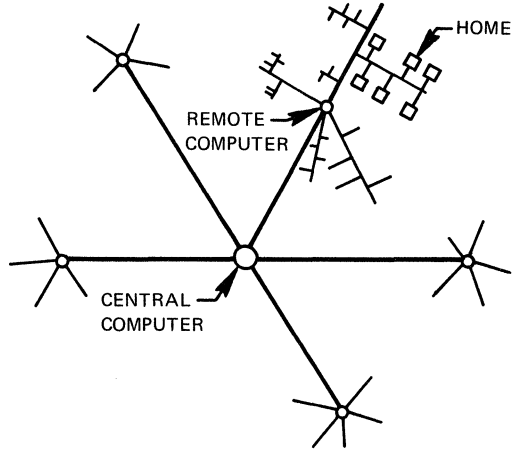


Fig. 15-9. TICCET Network/CATV System

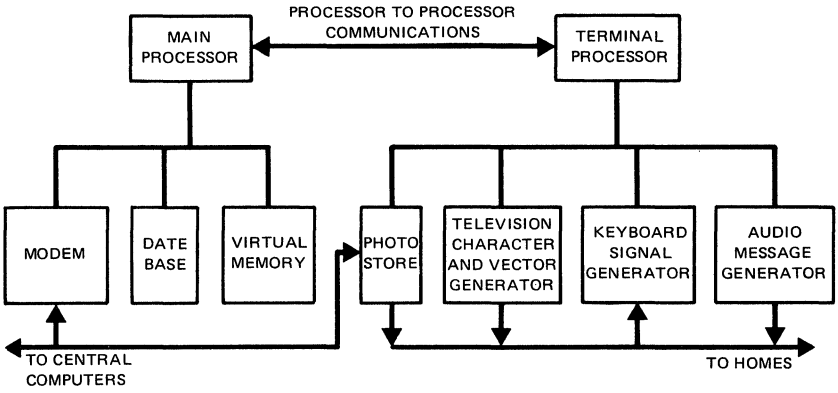


Fig. 15-10. TICCET Remote Computer

16. Video Graphics Performance Evaluation – Before and After Implementation

Thomas E. Bell
The RAND Corporation
Santa Monica, California

Other chapters have emphasized the performance of systems in terms of their value to humans. My topic is their performance in terms of the interactions of hardware and software. Specifically, I shall compare hardware/software performance evaluation before system implementation with evaluation after implementation. The earlier evaluation must be based on a series of unvalidated assumptions, whereas the later evaluation is based on reality – and provides the opportunity to judge the initial assumptions.

VIDEO GRAPHICS SYSTEM

The particular system under discussion is RAND's Video Graphics System (VGS) that Dr. Anderson discussed earlier. The user sits at a console with a number of input devices including, at least, a control box and a typewriter. Input from these is considered more important than input from optional input devices like The RAND Tablet.

Figure 16-1 shows the hardware configuration in use at the time of our analyses. A central digital-to-analog device is shared by all the terminals to reduce costs. The IBM 1800 serves as a communications switching and buffering device between the terminals and an array of service machines on which user programs run (currently, the 1800 communicates with an IBM 360/65, a PDP-10, and the Interface Message Processor of the ARPA network). All service machine and terminal I/O from the 1800 goes through a Special Purpose Multiplexer. This multiplexer has several interesting properties that influence the performance of the VGS. Among other things, it degrades the processing capability of the 1800 by 60 to 70 percent when it operates.

The software in the 1800 is somewhat unusual; no executive is employed. All software consists of one application program

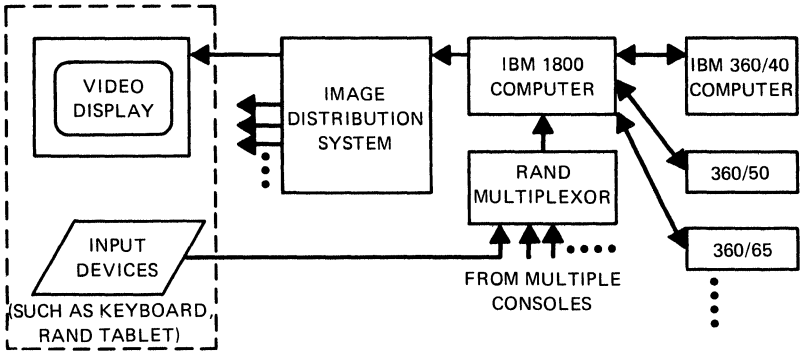


Fig. 16-1. Video Graphics System Hardware Configuration

designed to process interrupts coming from the service machines, the terminals, the special hardware, and from programmed interrupts.

BEFORE IMPLEMENTATION

During the design phase we were interested in projecting performance. About two months before the arrival of the hardware we developed the set of objectives shown in Fig. 16-2 (which is titled "potential areas of investigation" to distinguish its statements from the objectives of the system.) Some of these (e.g., numbers 1 and 2) were quite general, whereas others (e.g., numbers 4 and 5) were quite specific. We had one additional objective that we weren't clever (or brave) enough to specify in a meaningful, operational way; it involved a desire to learn enough about the system to develop some very simple, condensed statements about system performance. We had no technique to test ourselves when we felt we had learned enough, so this objective is not listed.

The objectives, combined with the lack of anything to measure, meant that our performance analysis technique was simulation. The results of our simulations of this forthcoming graphics system were analyzed with the use of an existing interactive graphics system. Figure 16-3 illustrates some of the output from that system when hard copy was requested. Several pictures were recorded, and the resulting hard copy frames were cut and pasted together to indicate VGS performance over a period of interest.

The output in Fig. 16-3 was created after much of the software was operating on equipment that, by that time, had been delivered.

WHEN THE DESCRIPTION EFFORT BEGAN, SEVERAL OF THE CHARACTERISTICS OF THE VIDEO GRAPHICS SYSTEM (VGS) SEEMED WORTHY OF INVESTIGATION BY A MODEL OF THE SYSTEM. SOME OF THESE ARE LISTED BELOW.

1. UNDER WHAT LOAD CONDITIONS WILL THE SYSTEM GIVE POOR RESPONSE? (IT MAY BE FEASIBLE TO ALTER THE LOAD BY USER EDUCATION AS WELL AS BY CHANGING CHARACTERISTICS OF SUCH SOFTWARE SUPPORT AS THE INTEGRATED GRAPHICS SYSTEM.)
2. WILL MESSAGES BE UNDULY DELAYED IN THE VMH SYSTEM IN THE 360s?
3. WILL CHANNEL CYCLE-STEALING SLOW THE 1800 CPU ENOUGH THAT INPUT DATA ARE LOST DUE TO DELAYS IN PROCESSING?
4. WILL A PING-PONG SYSTEM DECREASE RESPONSE TIME OF THE VGS?
5. WHAT WILL BE THE EFFECT OF THE 1800 WAITING AT INTERRUPT LEVEL FOUR WHILE BUFFERS ARE UNAVAILABLE FOR SERVICE MACHINE INPUT?
6. WHAT WILL BE THE EFFECT ON THE 1800 OF ONE SERVICE MACHINE BEING UNRESPONSIVE FOR A SHORT PERIOD?
7. WHAT PORTION OF SYSTEM CAPACITY DOES A TABLET TAKE? (IT MIGHT BE PROFITABLE TO DISABLE A TABLET THAT IS TEMPORARILY NOT IN USE, OR TO USE A KEYBOARD INSTEAD OF THE TABLET.)
8. HOW USEFUL WOULD MORE CORE BE IN THE 1800?
9. HOW USEFUL WOULD ANOTHER 1800 BE?

Fig. 16-2. Potential Areas of Investigation

The area enclosed by the box indicates a peculiar sequence of subprogram usage because one subprogram (numbered 8 in this display) was being invoked twice to perform a function that could be done with one invocation. Investigation indicated that, during card deck reproduction, an extra card had been inserted into the deck; detecting extraneous cards was an unanticipated result of the simulation effort.

In addition to the unexpected payoffs, we investigated the planned questions. One (number 7) inquired about the loading due to a single RAND Tablet. Figure 16-4 shows simulation predictions of the loading on the CPU (as a percentage of total capacity) required for various numbers of tablets. The diagram indicates the amount of activity at each level. Level 2 is a high priority interrupt level which performs processing caused by software interrupts from other levels. The high priority is used to ensure that a sequence of instructions is completed before other activity is allowed to intervene. Level 3 activity processes interrupts from the image distribution system; level 6 handles interrupts

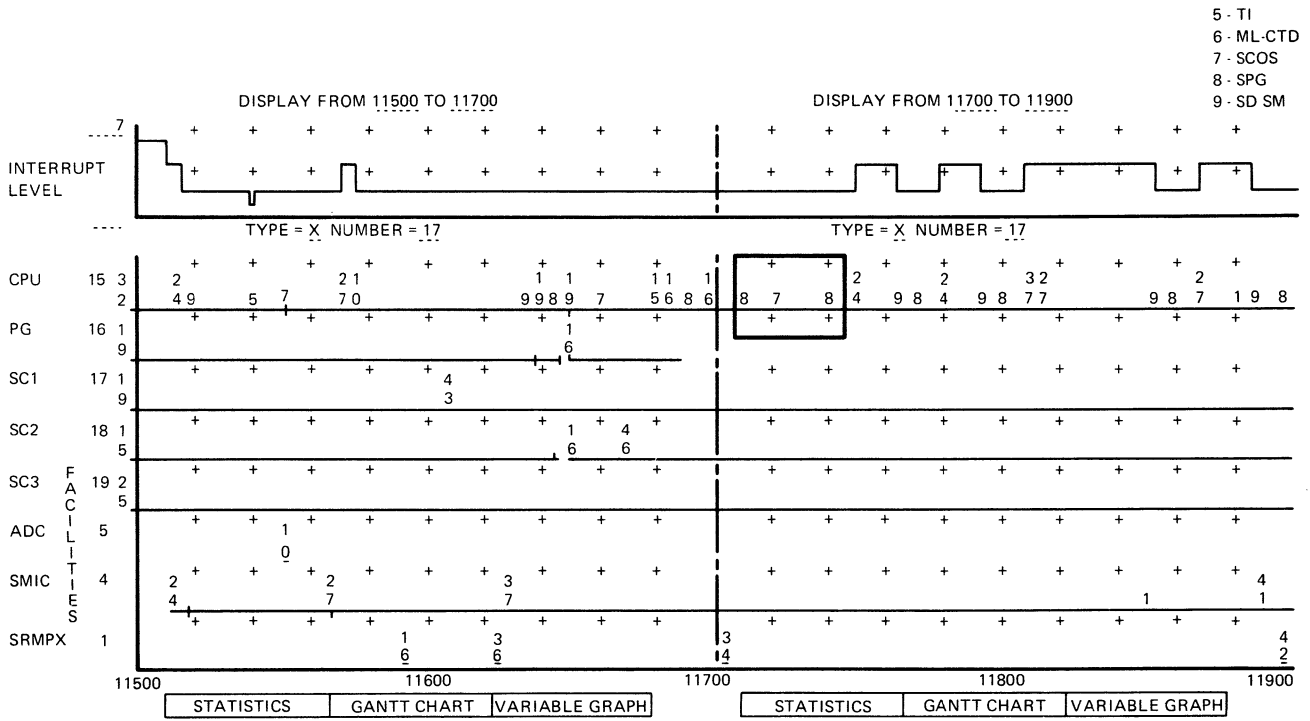


Fig. 16-3. Sample Simulated Output Used In Video Graphics System Design

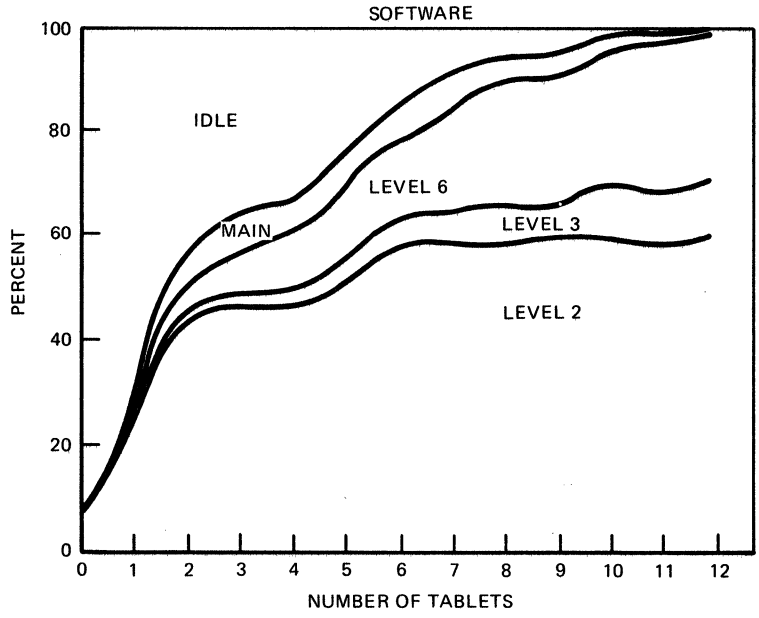


Fig. 16-4. Simulation Predictions of CPU Loading

from The RAND Tablets, and the Main level causes interrupts in level 2 to attempt initiating I/O to service machines. Total activity on the system goes to near-saturation when 12 tablets are on the system. This result, combined with results of early test runs on the system, motivated us to simulate an alternative in which part of the processing of Tablet interrupts is done in hardware. Figure 16-5 displays the projected situation.

This result appears alarming; unloading the CPU seems not to have been achieved. More intensive analysis, however, shows that the processing occurring in level 6 has decreased as hoped, and the capacity freed is now being used in the Main level and, through triggering in the Main level, in level 2. That is, activity that was "squeezed out" is once more occurring; additional Tablets could therefore be added and cause this additional activity to be deleted.

Two lessons were learned from this evaluation before implementation. First, the increased understanding of the system led us to decline to give simple statements about the "portion of system capacity" taken by a Tablet; the portion required varies with the number on the system. Therefore, Question 7 of Fig. 16-2 needed rephrasing. Second, we became very reluctant to quote any numbers about predicted system performance; important caveats can be easily forgotten when relationships are complex.

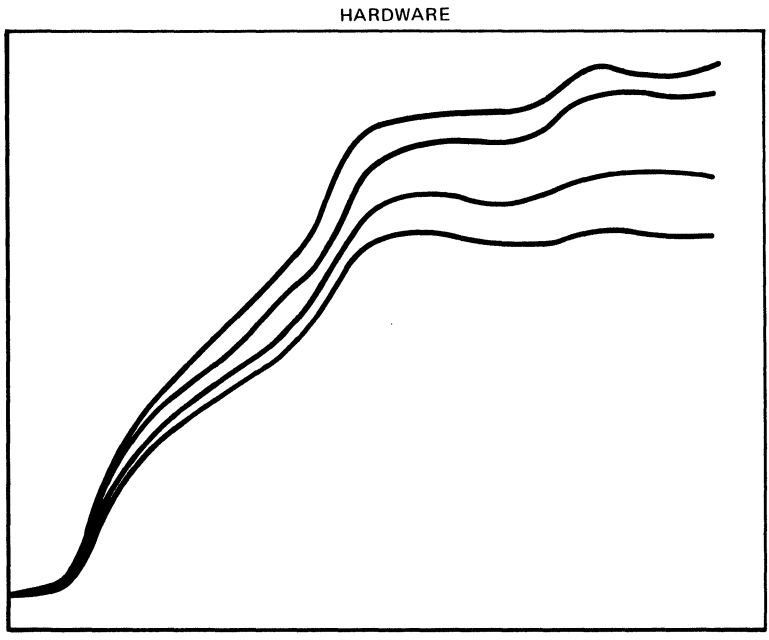


Fig. 16-5. Alternative CPU Loading Simulation

AFTER IMPLEMENTATION

The Video Graphics System was installed and began providing interactive graphics services to users immediately. After it had been in operation for several months, we had the opportunity to validate our simulation by measuring system performance with a hardware monitor. Wires were strung into the 1800 CPU to detect the status of various circuits. Each probe consisted of a connector feeding signals to an amplifier which transferred the amplified signals over the probe wire to a minicomputer based hardware monitor. Control over the monitor's functions was exercised through an online teletype. With this control we could examine the portion of time the 1800 was in each interrupt state under various conditions. The initial values from the monitor were validated through carefully designed tests using special program traps, an oscilloscope, and a signal generator. We found that the values from the monitor were accurate, so we could depend on results we obtained. Figure 16-6 shows the results from monitoring normal, uncontrolled use of the system over various periods at different times of the day. No useful information could be derived

MONITORED DATA

DATE	6/3	6/3	6/3	6/3	6/3	6/3	6/4	6/4	6/4	6/4	6/4	6/5	6/5	6/5	6/5	6/6
TIME	11:57a	2:40p	2:45p	2:50p	2:55p	3:00p	11:18a	2:24p	4:45p	4:50p	6:00p	8:47a	9:30a	11:23a	4:00p	3:00p
MINUTES REQUESTED*	30	1	1	1	1	100	30	30	1	1	150	10	1	30	30	5
*LEVEL 2	4	5	4			4	10	1	10	9	4	3		3	6	2
*LEVEL 3			0													
*LEVEL 4																
*LEVEL 5					0											
*LEVEL 6																
*LEVEL 8																0
*MAIN LEVEL	1	1			2	1	2	5	2	2	1	1		1		
*CPU BUSY	7	7	8	6	10	6	14	8	15	13	6	5	10	5	9	3
*CHANNEL BUSY FROM 360				0											0	
*CHANNEL BUSY TO 360				0											0	

*REQUESTED ON MONITOR; ACTUAL TIME IS APPROXIMATELY 2 1/2 TIMES REQUESTED
MONITORING NORMAL USE (TEST 19)

Figure 16-6. Monitoring Normal Use

from these data except that the system was quite variable and not highly loaded.

The problem with our initial runs was that we had no way to know how much load the users were causing, so we could not obtain any relationship between load and system response. We employed an artificial loading device to perform controlled experiments on the system in order to avoid the variability inherent in human activities. We had some initial problems with users running on the system during the time allocated to our controlled tests. (Users had no way of knowing for certain whether we were actually running tests and assumed we weren't.) This problem was solved by physically disconnecting all terminals not under our direct, physical control. The total load on the system then consisted of that introduced by the artificial loading box.

We examined our characterization of the interactions between interrupt levels by driving the CPU to saturation with pseudo-hardcopy characters (which used level 5). As can be observed from Fig. 16-7, the sort of activity we predicted is occurring. While a test checking for the situation in Fig. 16-4 could be suspect due to the relatively small portion of the system used by level 6, the ability of level 5 to "squeeze out" activity in the Main level, and thus in level 2, is very dramatic in Fig. 16-7.

The characterization of the level interactions was thus validated, but the absolute levels of activity were not examined in this test. We designed a test to give these values for the case of total software processing of RAND Tablet interrupts with the Tablet's pen up, and then down. Figure 16-8 shows the results, with the first column of values being those indicated by the simulation, and the second giving the ones observed on the system. The discrepancy was approximately a factor of 2 – often referred to as a 100 percent error. Much of this error was explained by some changes in software which had not been included in the simulation representation, but we suspected that this explanation was inadequate to account for the total amount of error. We went back to a very tightly controlled, very simple test to examine the cause of the discrepancy.

We returned to our box producing an artificial load and caused the system to process volumes of invalid characters. These invalid characters were handled very rapidly by the system since only 83 machine instructions were involved. Using the published, manufacturer supplied data on the 83 instructions, we computed the time that should be required in various cases. Figure 16-9 displays the results. The average computed value was within a range, but we could not determine the exact time because of potential sequencing which was dependent on the time to process. The values we suspected were most likely to have actually occurred are underlined, and errors (the amount of error we would state if the simulation deviated from the computed values) are also given. Some undefined effect (possibly an obscure

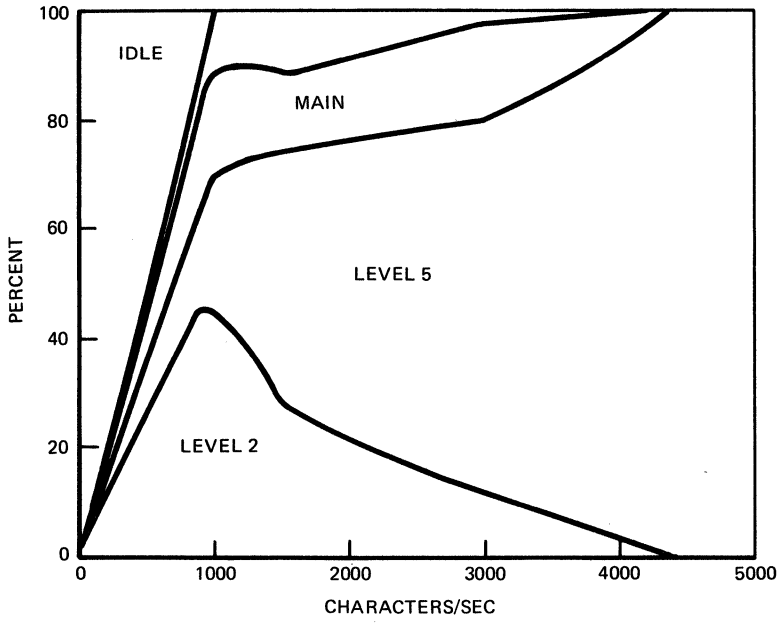


Fig. 16-7. Interaction Between Interrupt Levels

	1 UP	
LEVEL 2	12	27
LEVEL 6	0	2
MAIN	2	3
BUSY	15	32
	1 DOWN	
LEVEL 2	12	28
LEVEL 6	2	4
MAIN	2	3
BUSY	18	35

Fig. 16-8. Software DF

COMPUTED (μsec)	MEASURED (μsec)	"ERROR" (μsec)
155 - 239	317	33%
155 - 197	273	38%
155 - 183	220	41%

Fig. 16-9. Hard Input (83 Instructions).

option used in the software) causes the system to require more time than expected. (Intensive examination of the hardware showed that it was functioning normally.)

CONCLUSIONS

Four conclusions arise from this experience:

1. Detailed simulation analysis can be a very useful tool in designing and implementing online systems. Simulation can help discover unexpected characteristics of the system as well as answer predefined questions.

2. Simulation of alternatives can lead to improved gross characterizations (models) of the system through intensive analysis of differences.

3. Monitoring normal operations is not so useful as running controlled tests for simulation validation. Even if precise characteristics of loading can be obtained, the analyst must still separate out the parts of the simulation causing errors in a very complex situation.

4. Absolute errors may be incurred in simulations in spite of significant efforts to eliminate them. Unexpected interactions are likely in advanced online systems and will make simulation results inaccurate in, at least, absolute value.

17. Performance Capabilities of Hardware Systems

Cay Weitzman
System Development Corporation
Santa Monica, California

This chapter aims to provide a survey of present hardware performance and capabilities. The information is abstracted from several recent hardware evaluations. Such evaluations are undertaken by SDC on a continuing basis.

The survey will encompass computer architecture in the real-time environment, where things stand today in terms of peripherals, and some points on reliability and trends.

Today, the general trend in computers is toward modularity: modularity in terms of CPUs, memory, and I/O devices – all elements that fit into the computer. The emphasis is on minicomputers, where modularity is even more prevalent. An example of a very typical minicomputer that has these features is the CDC 11 which comes in various configurations and each configuration can be built up or down in just about any way desired.

In terms of register blocks, read-only-memory, scratchpad, cache memory, and virtual memory, particularly the latter one, we find it today in the biggest computers and particularly the virtual memory used by IBM in its latest 370 series. The associative memory is not here except for some specialized machines (STARAN and PEPE), but if virtual memory is desectored small enough, it will eventually wind up as an associative memory.

Computer Architecture

One important feature in computer architecture in the real-time environment is the interrupt structure. A large number of real-time computers today have a multilevel hardware interrupt structure. They provide internal/external interrupt either by device source, program, or through buffered I/O. The level changes are of particular importance in real-time systems – command and control systems – where one must change quickly

from one interrupt level to another. In a typical medium-scale command and control computer, a computer used in a real-time environment, this could vary as much as five to 100 or 500 microseconds. This "overhead" has a great impact on the total response time of the system.

Many of the presently available computers have both direct memory access and buffered I/O channels and data also can be moved to and from memory under program control. Many systems have a wide choice of I/O interface, either serial or parallel lines that one can input data to the system. Other features are multiplex I/O, asynchronous or synchronous lines, analog interfaces, such as A/D, D/A and synchro-to-digital which are also used in process control applications.

A very interesting trend in interfacing various low-speed peripherals is the use of USAC II control characters. In other words, instead of using separate control lines, the peripherals are controlled by control characters which gives one greater flexibility in inputting and outputting from the computer.

Someone has said minicomputer manufacturers are becoming more like Detroit every day. There seem to be more and more variations on a lesser number of machines. The typical trend in the minicomputer area is, of course, lower cost. Everything is getting cheaper and cheaper, faster and faster, and many of the new systems introduced in the last few years are microprogrammable. Interdata was one of the first commercially available microprogrammable minicomputers, but there are a whole host of them now.

In general, one can conclude that the cost decreases in all the minicomputers today are due to minor amendments to details across the line rather than from significant technological breakthroughs.

Going to military machines, the most typical feature of military machines is the advanced packaging concept and microminiaturization. Medium- and large-scale military computers have a more advanced architecture, also in terms of building diagnostic capability and this of course gives a short mean time to repair.

The commercial computers today are more sophisticated in terms of firmware control, virtual memory, integrated controllers, powerful I/O processors and both LSI and other types of memories. Emerging architectures such as associative machines or computers based on associative memories, such as PEPE from Bell Labs, on pipeline computers such as the ASC from TI (Texas Instruments) and STAR from CDC, are machines that although not available to the commercial user represent trends that, I think, will have very strong impact on future computer design.

One issue raised by many people is the mini versus the large computer. Is the medium-sized computer going to disappear? Will there be a lot of minis or a central multiprocessor or a large computer? There again I think a trend has really not crystallized

yet. Almost every day there is a new medium-sized computer. The latest one announced is the Univac 9700.

I have tried to assemble various computers in terms of word size (Fig. 17-1). There are, of course, other ways of ordering them – in terms of throughput or number of instructions per time unit. Here though, we have word size versus cost. The minis today are creeping down toward $1000. Going in the other direction, we have the midi, which is not really a small computer, although that is debatable. The midi is a large mini. Typical midis are the Honeywell DDP-516, the SDS Sigma 3, the IBM 1800, and the CDC 1700. The small computer is something like a 360/20 or 30. The next level is the medium-sized computer. Typical medium-sized real-time systems are the PDP-10, the XDS Sigma 5, the IBM 360/44, the SEL/86, the Honeywell 632, and the Univac 418.

A very similar pattern exists in the military field (Fig. 17-2). All these are ruggedized minicomputers. There are minis such as the Rolm 1601 which is a ruggedized Data General Nova, and others such as the DDP-516 which has been designed in a ruggedized version and also a miniaturized version (HDC 601). The HDC-601 is a small airborne computer, but it has the characteristics of the commercial DDP-516.

At the other end of the spectrum are the large command and control computers, the Hughes 440 which is a new system, the Litton L-3050, the TACFIRE computer, and the Univac AN/VYK7 built for the Navy (AEGIS). Marconi in England has built a military minicomputer called the 920M Myriad and Hitachi in Japan built one which is used as an airborne navigational computer.

Finally, the very large computers in the multimillion dollar range are really not militarized per se. There are militarized versions such as the PEPE, the ADC, and the Univac CLC.

The five year trend is shown in Fig. 17-3.

Figure 17-4 is a summary of various military computer categories. It shows the memory, CPU, and I/O differences among these various groups and it also gives an indication of what type and range of peripherals are available to the military computer system designer. The microminis have almost no peripherals available or even a peripheral interface capability; one must go to a fairly large-size computer to get adequate peripherals. And even here the peripherals one wants on a system are usually picked from a variety of manufacturers and are sometimes not compatible so one may have problems with the interface.

The micromini, I think, is a very interesting design concept, based on LSI. We certainly have LSI computers here today and there are quite a few of them on the military market – the Control Data 469, the Bunker Ramo BR-1018, the Garrett AiResearch Adapt series and the Autonetics D200 series. They are probably the forerunners for the commercial market, which will use very small, compact, ruggedized computers in cars, trains, wherever some kind of control is needed.

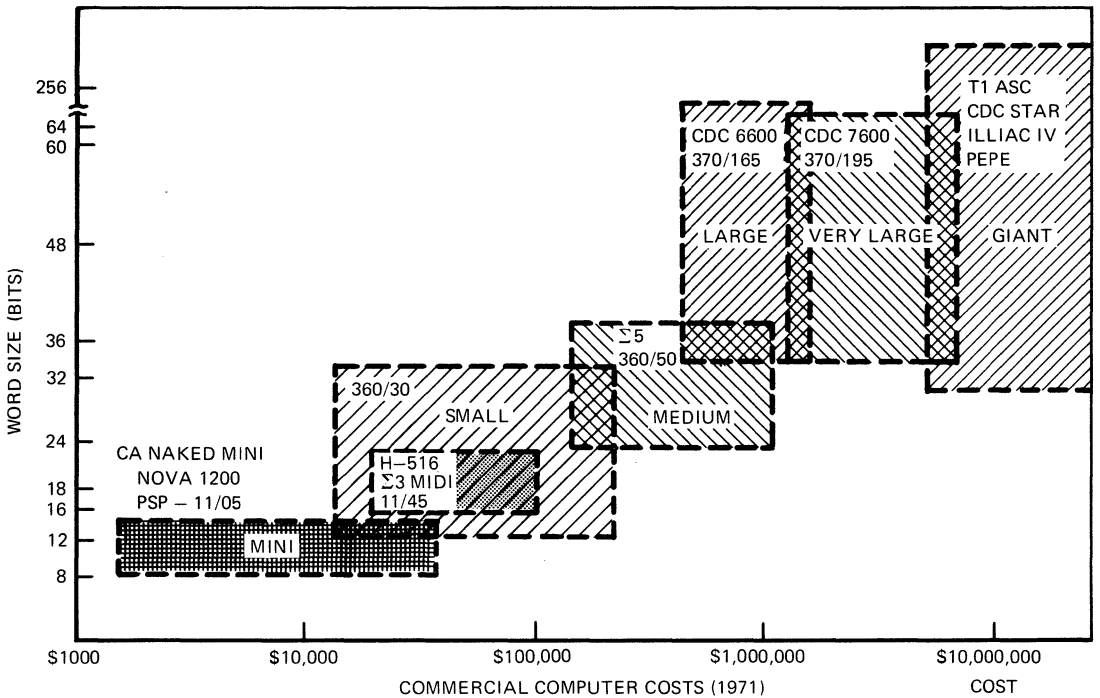


Fig. 17-1. Commercial Computer Costs (1971)

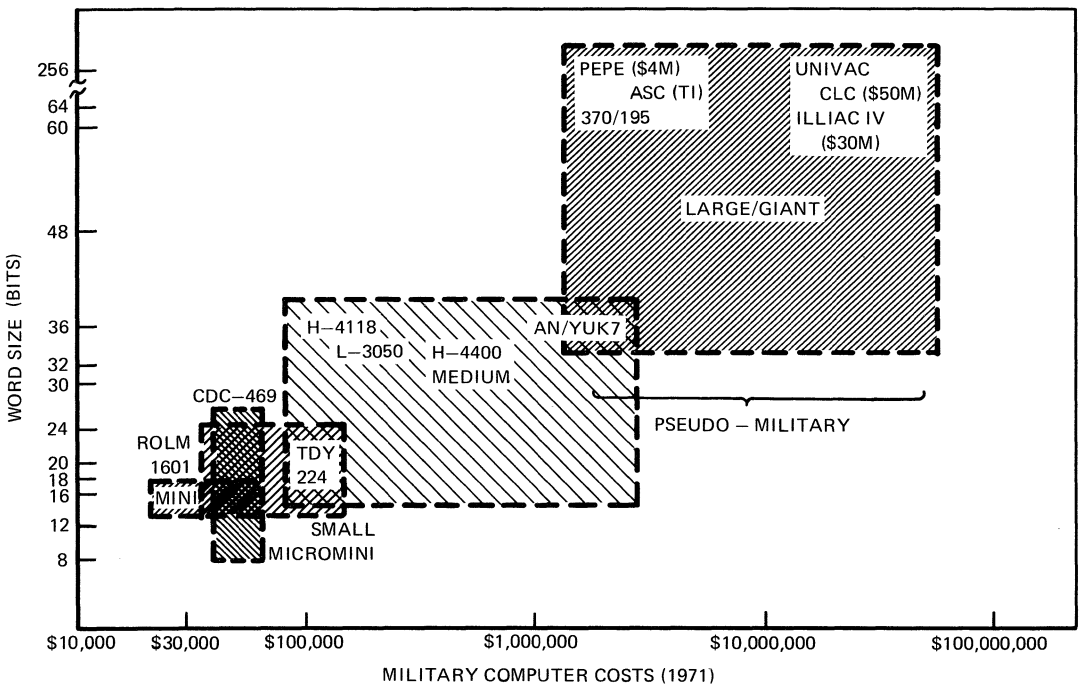


Fig. 17-2. Military Computer Costs (1971)

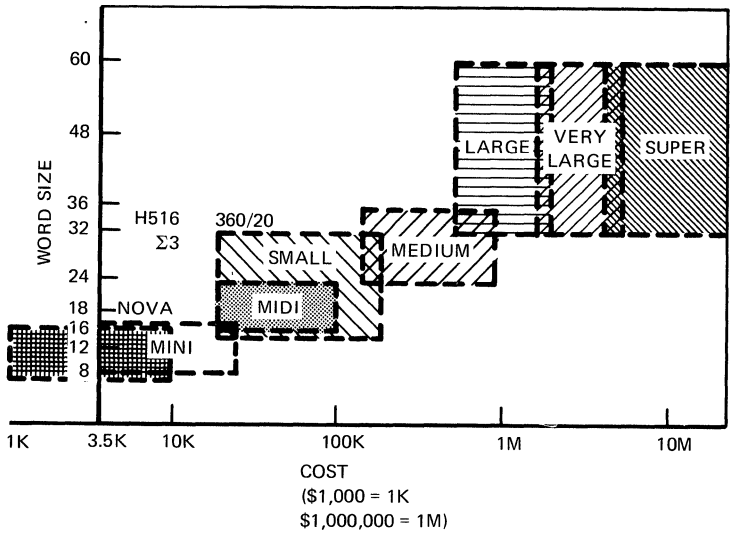


Fig. 17-3. The Five Year Trend In Computer Costs

The trend in new system design will be very much influenced by high-speed data links. Today most systems – whether they use teletype terminals at 110 bits per second or the IBM counterpart, the 2741, at 134.5 bits per second or high-speed 2400 to 4800 bps synchronous lines – have a remote I/O capability. We are going for even higher data rates and some systems use 9600 bits per second data rates or beyond. In spite of this, most computer systems have relatively limited data communication capabilities. They are not designed to handle very high data rates but a big change is expected there, particularly due to new facilities that are going to be available. Such facilities include T1 lines with voice channels going up to 1.5 million bits per second and eventually the T2 which will be used for picturephone transmission and has a 500-mile range. New data sets such as the 303 or the 306 will have about a half-million bits per second data transmission capability.

Today most military systems are locked up with AT&T and their long distance transmission lines. I believe firmly that the near future availability of higher data rate transmission links such as T1 is going to have a very strong impact on architectural change in computers in the next few years.

COMPUTER FEATURES OR CHARACTERISTICS	MILITARY COMPUTER CATEGORIES			MEDIUM AND LARGE (MULTIPROCESSORS)
	MICROMINI	MINI	SMALL	
MEMORY				
CYCLE TIME	1.0 TO 5.0 μSEC	0.9 TO 2.6 μSEC	1.0 TO 3.0 μSEC	1.0 TO 2.0 μSEC
WORD SIZE	12 TO 24 BITS	16 TO 18 BITS	16 TO 24 BITS	32 BITS
STORAGE TYPE	CORE, PLATED WIRE, MOS/LSI	CORE	CORE, LSI	CORE
MEMORY SIZE	1k TO 13k WORDS	1k TO 65k WORDS	2k TO 13k WORDS	UP TO 256k WORDS
CPU				
NUMBER OF INSTRUCTIONS	30 TO 70	50 TO 100	20 TO 120	60 TO 180
REGISTER	NONE OR VERY LIMITED	TYPICALLY 2	LIMITED	LARGE NUMBER
ADDRESSING MODES	TWO OR LESS	THREE OR LESS	SEVERAL	LARGE NUMBER
OTHER				LARGE NUMBER OF MACROS
I/O				
DMA	250 TO 400 kHz	300 kHz TO 1 mHz	300 TO 1 mHz	500 kHz TO 1.4 mHz
OTHER			SOME WITH A/D, D/A, S/D CONVERTERS	SERIAL, PARALLEL UNDER CPU CONTROL
				LARGE NUMBER OF I/O PROCESSORS
PERIPHERALS	LIMITED OR NONE	LARGE NUMBER, MOSTLY COMMERCIAL, NON-MIL SPEC'ED	LIMITED	ADEQUATE, SOME COMMERCIAL
PACKAGING				
COOLING	CONDUCTION	GENERALLY FAN COOLED	CONDUCTION	FORCED AIR, LIQUID, CONDUCTION
SIZE	10 TO 450 CU. IN.	0.5 TO 5.0 CU. FT.	0.5 TO 1.0 CU. FT.	6.0 TO 25.0 CU. FT.
WEIGHT	0.5 TO 10 LBS	40 TO 60 LBS	MORE THAN 10 LBS	50 TO 700 LBS
RELIABILITY				
MTBF	7,000 TO 25,000 HRS MIL SPEC	1,000 TO 10,000 HRS SOME MIL SPEC	2,000 HRS TYPICAL MIL SPEC	1,000 HRS TYPICAL MIL SPEC
COST; MINIMUM CONFIGURATION (CPU MEMORY BLOCK, I/O, POWER SUPPLY)	$50,000 – $80,000	$20,000 – $50,000	$40,000 – $150,000	$120,000 – $300,000

Fig. 17-4. Summary of Military Computer Categories

Peripherals

As for peripherals, the lack of high-speed computer output devices, except for computer-output-microfilm equipment (COM) and some other exotic devices like the laster printer (still under development) and the more conventional peripherals, such as line printers, magnetic tape and card or paper tape punches still dominate the user market. Therefore, I am going to spend a few minutes discussing conventional hardware and the changing trends in peripherals.

One very interesting phenomenon is the cassette tape, currently quite popular. There are more and more different types of cassette tape units and most of them seem to be centered around the Phillips cassette. This unit hasn't really been around long enough to show a record of success in terms of low error rates. Some problems exist with these units if one wishes to record very large continuous blocks of data. An important feature is read-after-write to reduce errors.

Several low cost printers on the market today use either matrix or helix type printing. The price is now in the $3000 to $5000 range.

The alphanumeric CRT terminal also has been coming down rapidly in price. Just three or five years ago, a $10, 000 interactive CRT was common and today it costs $2000 to $3000.

Only in teletype is there little activity. Few contenders can really compete with the $600 to $700 teletype. Most people seem to be using them widely. Of the military peripherals, I think the most significant activity is in display. There are a large number of new types of displays, most of them based on the CRT. These displays are designed both in the U. S. and in countries such as Italy, France, Germany, and Japan. Many of these are quite sophisticated in that they allow a large amount of user interaction, displaying PPI display together with synthetic data, some with bandwidth compression and color capability. Many of these systems are also coming down in price. Many of them, as a matter of fact, are off-the-shelf devices – they can be ordered and integrated with many types of computers.

Figure 17-5 indicates where we are in terms of peripherals today, based on throughput rate and characters per second. Everything has been brought to a common denominator, characters per second, going from 10 characters per second up to 10 million characters per second. And, as the reader can see, the Teletype is at the far left end of the low-speed devices followed by line printers, card readers, and paper tapes. On the top line is COM equipment. The really significant message in this chart is the large discrepancy in data rates between I/O peripherals and the actual CPU or memory devices; the only ones getting close to the disk or drum speed are the computer output microfilm devices and the magnetic tape, which really indicates that we need some higher speed peripherals than those now available.

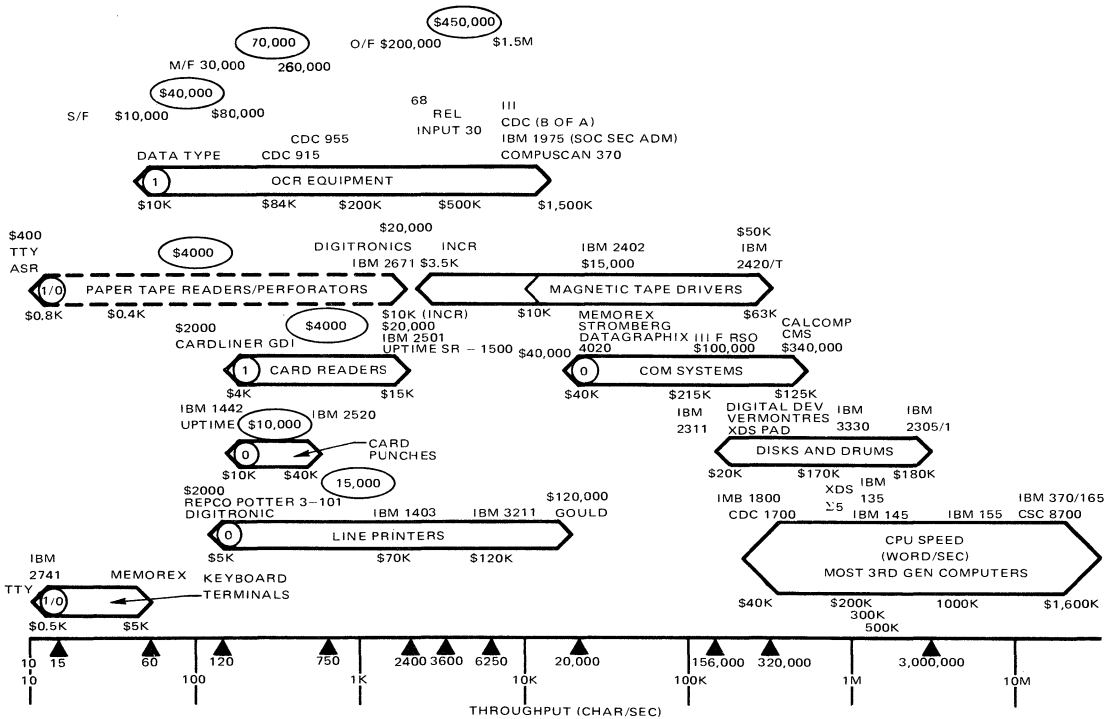


Fig. 17-5. Current Peripheral Capabilities Based on Throughput and Character per Second

In military systems, the applications and levels of reliability are closely related. Space or missile computers installed in aircraft require very high reliability. It would be hard to get up there to fix them when they fail. Many of the new commercial minis discussed earlier have also been converted to ruggedized or militarized systems. A major advantage in using these minis in a military application is the savings in the basic software development cost. The systems have been around for some time and there is a lot of experience with them.

Self-Test and Repair Machines for Missile and Space Environments

Several projects are going on right now to develop self-test and repair systems. Burroughs is developing a system (still not announced officially) called the Model D. There are computers which require function monitoring in terms of hardware and error-correction circuits with hardware monitoring and triggering for later action. One example is the STAR, the JPL self-test-and-repair computer.

Finally, I would like to examine several trends. I have been doing some work with various military computers, such as the Litton L-3050 and the Hughes H-4118. Many of these systems can be checked out with hand-held card testers. The tester is plugged onto the back of the card and the circuit boards can be checked out quickly. This is still a quite limited capability. There are only X numbers of cards that can be checked out this way. Certain kinds of functions don't lend themselves to being checked with this tester. This approach is, however, extremely powerful in a field environment.

Many things have been said about LSI. Supposedly, it is the solution to most of our problems. Computers can be designed on the side, so to speak. Just using LSI technology, one can quickly design the function to do whatever desired. I don't believe we are there yet. Several computers, like the Control Data Alpha, use MSI/LSI. Building these units involves a high cost. The general idea is that if something fails the board with the LSI chips, it is pulled out and thrown away. That is quite an expensive way to go. The more LSI on the board, the higher the cost of the board and the more expensive to repair or maintain.

Although some systems use welded modules, I feel that the trend is away from that. You see more flow soldered or dual-in-line plug-in boards for changing or replacing some of the subcomponents. A very pronounced trend in military computer design is toward a short mean time to repair capability. With built-in diagnostics, problems are sometimes located more quickly than with the hand-held tester. One simply can work through the computer itself. Other protective measures, such as use of fail-safe capability, is increasing in both military and commercial computers. All contribute to system reliability.

18. Toward Natural Man-Machine Dialogue

M. I. Bernstein

System Development Corporation
Santa Monica, California

The goal of this work is to provide man-machine communication that is as natural as the dialogue between two colleagues. The phrase "between two colleagues" is chosen with care, for we do not mean to imply that we can make communication between man and computer as free and arbitrary as that between two randomly chosen individuals, for that problem is not solvable in the short run. What we are trying to provide is the framework within which the machine may be treated as a colleague by specifying a context that properly bounds the domain of discourse.

Considering that communication implying understanding implies intelligence, one is tempted to class this work as a manifestation of artificial intelligence. The appropriateness or accuracy of such classification is irrelevant, particularly if we are able to reach our stated goal.

Three projects that we collectively call the Natural Input/Output task are attacking three aspects of natural man-machine communication in parallel. The English I/O Project is working toward a natural English data management system; the Voice I/O Project is attempting to solve the speech understanding problem, also using data management as a vehicle; and the Graphic I/O Project is providing capability to communicate with a computer in printed or written natural scientific notation. The necessity of the latter is obvious to anyone who has attempted to linearize or verbalize a complex mathematical expression.

ENGLISH I/O

The CONVERSE Project is developing a natural English data management system. Interrogatory, declarative, or imperative English sentences are input to the program by keyboard. These are parsed, in a fashion similar to that which was at one time

taught in grammar school, but with more rigor and depth using the principles of transformational grammar and techniques developed in computational linguistics. Meaning or the semantic content is extracted from the sentence with the combined use of the dictionary, fact file, and concept net. The details of the method have been documented and are inappropriate here.

The parsed result is cast into a formal intermediate language (IL) that is a variant of the calculus. The IL forms the basis for the generation of a procedure that is executed by the data management portion of the system, producing the specified result: the answer to a question, or modification to the data base content. Since there are limitations on both subject matter and grammatical freedom, the parsing attempt may result in a request for the user to define a term, choose among the alternative meanings of an ambiguous sentence or reject the sentence outright.

Some samples of sentences acceptable to CONVERSE are reproduced below. The subject matter in the data base in this set of examples is about census information from cities in California and New York.

What is the population of each city?

Name pop*60

Alameda 65000
Concord 36208
Downey 82505
Los*Angeles 2479015
Lynwood 31614
More ?

Y
Modesto 36585
Oakland 367548
San*Diego 573224
San*Francisco 740316
Albany 129726
More?

Y
Buffalo 532759
New*York 7781984
Yonkers 190634
No more

(The population of all 135 cities in the data base could be obtained with patience.)

Is it Yonkers that isn't located in California?

Yes.

Which cities are not located in California? Name

Albany
Buffalo
New*York
Yonkers
No more

(Circumlocution in the first example and negation are properly treated.)

What are the northeastern cities with the most and least land area?

The one thing with the maximum of land*area* 60 for northeastern things has the city is Buffalo

The one thing with the minimum of the land*area*60 for northeastern things has the city is Yonkers

This is how ambiguous sentences are treated.

What's the MFI of NY?

2 parsing

2 interpretations

N Search Request

1 What is the median*family*income*59 for New York?

2 What is the median*family*income*59 for New*York* state?

Type: Answer N!

Answer 1!

The median-family*income*59 for New York is 6091

Answer 2!

The median*family*income*59 for New*York*state is not known

The last is an example of the effect of a declarative sentence on the data base content.

Present work on the project is directed toward extending the scope of acceptable language forms, simplifying the creation and update of data bases, and generally imbuing the program with more "intelligence."

VOICE I/O

The goal of the Voice I/O Project is to demonstrate the feasibility of a speech understanding program. The vehicle for the demonstration is a vocal data management system. The choice was made with malice aforethought. The foundation provided by the English I/O project will materially help in reaching the goal in the shortest possible time, between three and five years.

At present, a laboratory is in place and we have replicated the speech recognition program developed at Stanford by Vicens and Reddy, the best program to date for recognizing single words or phrases. But the goal is to have the system understand continuous speech. Note that in one case there is recognition and in the other there is understanding the distinction that in the latter case, there is no need to reproduce with any accuracy the input, only that the reaction or response be correct or acceptable.

We believe that if properly approached the problem is tractable and we believe we have a proper approach. The basis of that approach is called Predictive Linguistic Constraints (PLC). Figure 18-1 is a data flow (as opposed to control flow) of the PLC model upon which the implementation of the program rests.

The basic concept is that in any area of bounded behavior, such as using a data management system, that the speaker will be in a narrow bounded context a very high percentage of the time, and that the loss of predictability will occur when the speaker changes context. Since the number of contexts to which he may change is limited, the transitional case leads back to the well bounded predictable case in almost all instances. In the few cases when the program gets "lost," there is a human available, the speaker, to bail it out. In essence, it is not much different from the way two people interact.

GRAPHIC I/O

The primary goal of the Graphic I/O Project from its inception has been to provide natural communication with the computer through hand-drawn, -printed or -written input. As a result, we have not only developed software toward this end, but hardware in the form of a unique data tablet graphics console that is nearly as natural to use as pencils and paper. It incorporates a projection CRT, a data tablet, and the necessary optical system to provide a single interactive surface. It is partly true that it is done with mirrors.

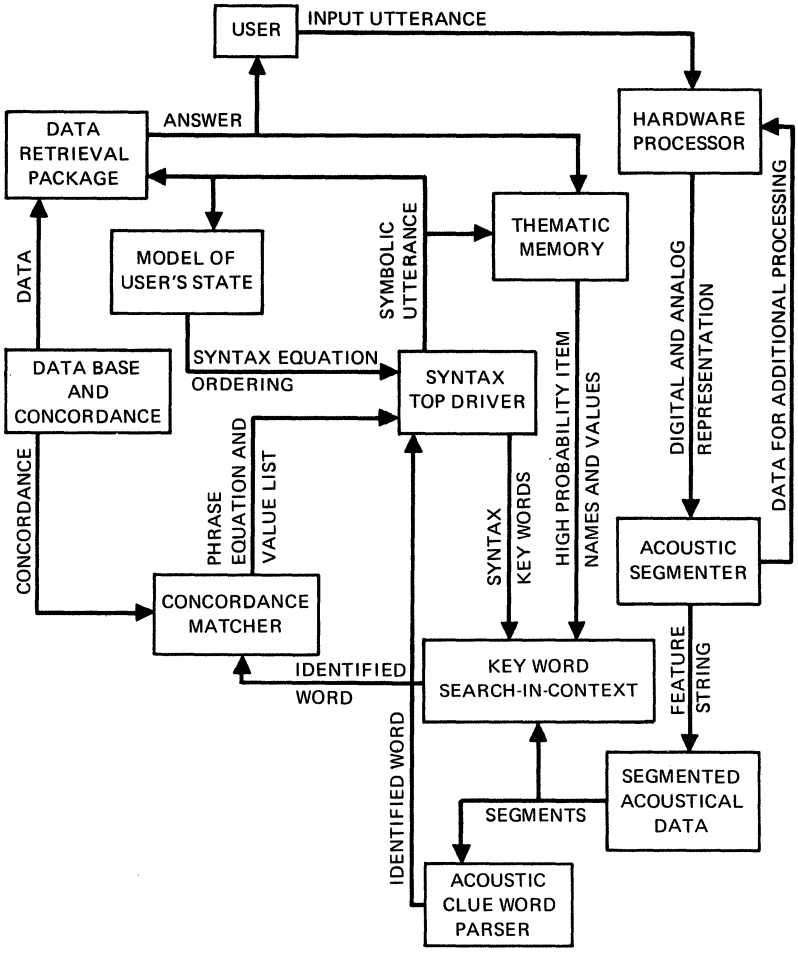


Fig. 18-1. Data Flow of PLC Model

The initial problem that was attacked and solved was the creation of a character recognition program that could accept a large variety of symbols from a large and diverse set of users. We believe that our character recognition program is more than adequate for the kinds of usage we envision in the future. Next, a program for accepting mathematical notation was implemented. The necessity for a companion program that converted the internal linearized computer representation of an expression back to its displayable two-dimensional form became clearly obvious. Given the ability to hand-print mathematic expressions into a computer and have them redisplayed in well-spaced textbooklike form is all

well and good, but certainly not an end in itself. We, therefore, created a computational facility that we call TAM, or The Assistant Mathematician. A few of its facilities are illustrated in Figs. 18-2 and 18-3.

Figure 18-2a shows the definition of the function V(h). The two versions represent the input through the recognizer, the lower centered version, and the formatted output, up and to the left of the first. Figure 18-2b shows the addition of a new definition T(h) which is a function of V(h). Figure 18-2c is a request to compute the value of T(150).

Figure 18-2d is a continuation of the sequence. The user has given a value the variable or constant called "R". The program reminds him, the user complies in Fig. 18-2e, the program again prompts (2f), asking for g and when it is satisfied, supplies the value for T(150) in Fig. 18-2g.

Figure 18-3 shows how iteration over a set of values is specified; Fig. 18-3a shows the input; Fig. 18-3b, the result.

The next logical step for this project is to select an area of specific applicability, bound its context much as the other two projects have done, and see if that can provide the vessel for creating a natural system that contains no surprises for the user and within his or its context always does what is expected without the need for specifying a great deal of the obvious.

FUTURE SYSTEMS

What does the future hold? It is not beyond the realm of the possible that before 1980 there will be systems available with which one will use the most natural form of input and output with the computer that fits the situation. One will speak those things normally spoken and draw pictures and write equations for those things normally drawn and written. The computer's response will be the spoken word, perhaps a song where appropriate, pictures, graphs, and charts, and even the printed word. A great deal of work going on in parallel with our own makes it clear that it is not a question of whether such systems will be achieved, but rather when.

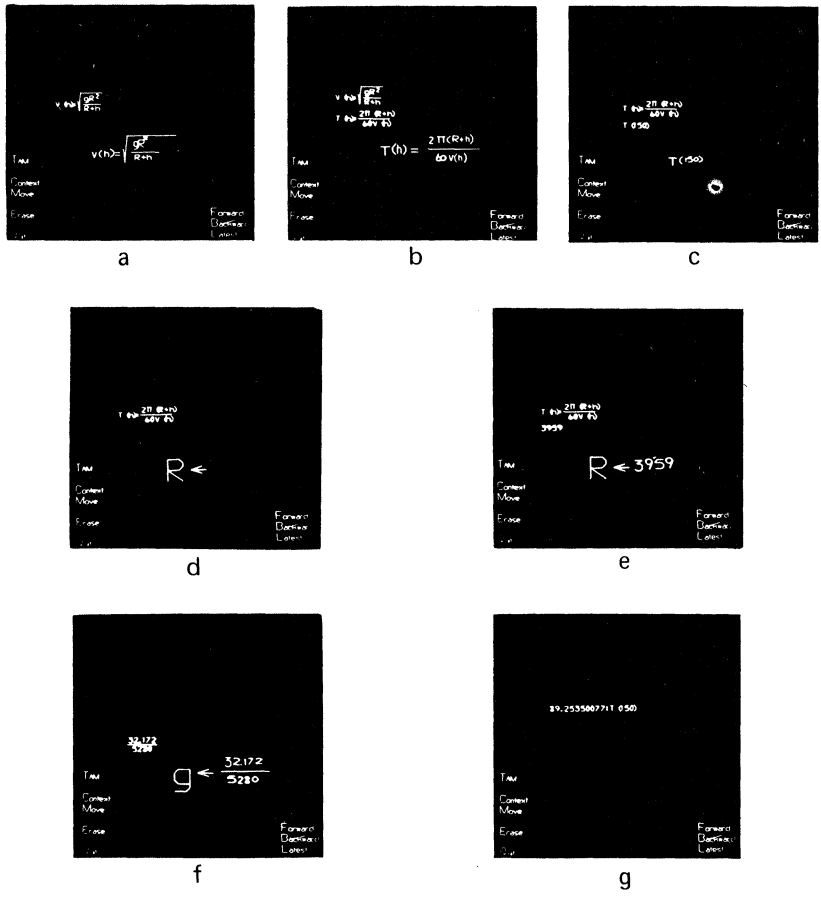


Fig. 18-2. Functions

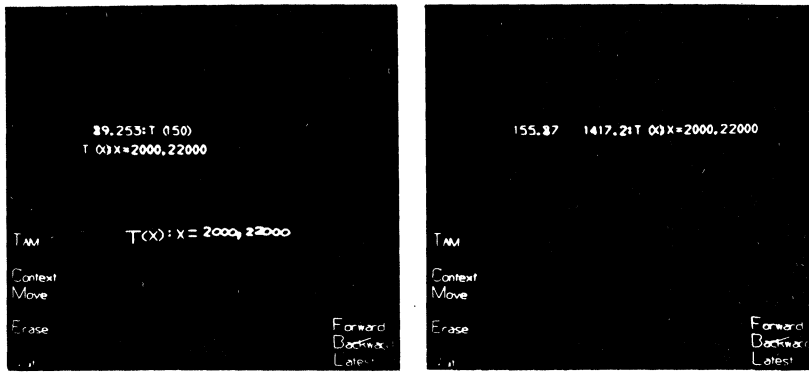


Fig. 18-3. Iteration Statement

19. REL: A System Designed for the Dynamic Environment

Bozena Henisz Dostert

California Institute of Technology
Pasadena, California

REL stands for Rapidly Extensible Language System, which is being developed at the California Institute of Technology. REL is an integrated software system designed to facilitate conversational interaction with the computer, especially on the part of those working with dynamic, highly interrelated data, in situations where the data is not only to be accessed but also to be manipulated in various ways by the user to suit his specific needs. In such situations, the user must be able to work with his data in a natural manner, through a language that is natural to him and best suited to his task, a language that allows him to analyze the data in a most facile and meaningful way. An essential ingredient of such a language is its extensibility, the ability to define and redefine terms so as to find the essential interrelationships in the data.

Most current data management and analysis systems are built around the idea of the total management information system. In such systems, data are collected from all over a large organization, stored in a large and all-encompassing data base, and made available to higher levels of management through statistical analysis routines or report generators. To attain the necessary levels of efficiency, the operation of such a system must be centralized; to remain sufficiently stable to be useful to the management, the content and format of the material must be closely regulated and under the control of the information system operation.

But there are other kinds and uses of data in organizations, whether they be a research center, an industrial setup, or a military headquarters. Each research team or staff office has its own information files which are used constantly as an ongoing part of the work. These may be records and results of a current series of experiments, or the data and models the team is working with in putting together a special study, or working files of raw

material relating to ongoing research, or records on alternative budgets and planning charts used in the preparation of a new program proposal. In all such cases, the research team or staff office is directly involved in gathering and maintaining this material, in making day to day decisions on its contents, formats and file organizations. Such material is not appropriate for the master file of the larger or outer organization and it is far too dynamic in all its dimensions for standardization. Further, those who develop and use such materials would not think of giving up control over them, for they are in a real sense the stuff and substance of their ongoing work. These are the dynamic, working files that constitute the essence of research and staff operations.

In order to build a system responsive to user needs in such dynamic environments, we have been especially attentive to two characteristics of the work of individual users or groups who analyze different aspects of a body of data much of which may be common to several individuals or groups. First, they need to deal with their data in an individualized manner, to dissect it in new ways, to test even far-fetched hypotheses, to build up their terminology in order to deal with the data most efficiently. Second, they need to communicate with each other's data, consult, and benefit from each other's analyses.

What, then, should be the characteristics of an information system which aims to support the working files, and habits, of many staff offices and research teams in dynamic environments?

First, such a system has to have the capacity to handle highly interrelated and time-oriented data. It must allow individual queries and analyses along unanticipated avenues and allow for the tracing of complex interrelationships. The very essence of research and staff studies lies in the search for new interrelationships, following of clues, and even guesses, tracing of implications and clarifications of emerging patterns. Thus, a system designed for supporting such operations must facilitate innovative, unprogrammed exploration of the data.

Second, in such an information system, communication between the user and his data must be in a language natural to him and tailored to his needs. On the one hand, this requirement calls for man/machine languages built on the syntax of natural language. On the other hand, the vocabulary and idioms of such languages should be those of the working teams or individuals, they should reflect the idiosyncratic dialect built around the concepts and interrelationships relevant to their work.

Third, such a system must be able to accept new terms and new data as well as new definitions of functions and relationships in the process of the ongoing use of the system; and it must incorporate these language and data base extensions for immediate use and further extensions. The changing, dynamic character is essential to the work of the staff, since the modifications in the data base and the concomittant modifications of the language reflect

the staff's maintenance of the relevance of the data, and so reflect how well they are doing their job.

Fourth, such an information system must provide enormous flexibility, which allows for a variety of user language/data packages, individually tailored to specific needs, and provides a facility for the intercommunication of such specialized packages. It must allow the addition of new language/data packages and new algorithms and the employment of a wide variety of data structures.

Fifth, such a system must have good response times.

The REL system is designed to fulfill all of the above requirements. It has already stood the test of users with needs such as were discussed above. This experimental REL system was in operation in the spring and summer of 1970. We are now developing a fully operational prototype. This prototype will be a fully interactive, multiprogrammed system by which a number of researchers can communicate directly with their data and models in a conversational way, time-sharing the computer facilities. This prototype system should be in operation a year from now on an IBM 370/135 in a test environment. We plan for a debugged, evaluated and documented system by Fall, 1973.

In this paper, only some of the outstanding features of the system are discussed and illustrated; namely:

1. its ability to handle interrelated and time-oriented data;

2. provision for communication with data in natural language, tailored to user needs, with emphasis upon ordinary English;

3. the extensional facility, which allows for the modification of data through definitions of new terms and relationships as part of the user's ongoing work with the system.

The third point receives special emphasis, since its discussion and illustration also serve to bring out the other features.

First, however, the general architecture of the system needs a brief presentation. It is more fully discussed in references 1, 2, 7.

THE REL SYSTEM DESIGN

The REL system has three main parts. As Fig. 19-1 shows, the three main parts are:

1. the operating system, which manages the simultaneous use of the system from a number of terminals and handles all input/output from peripheral storage;

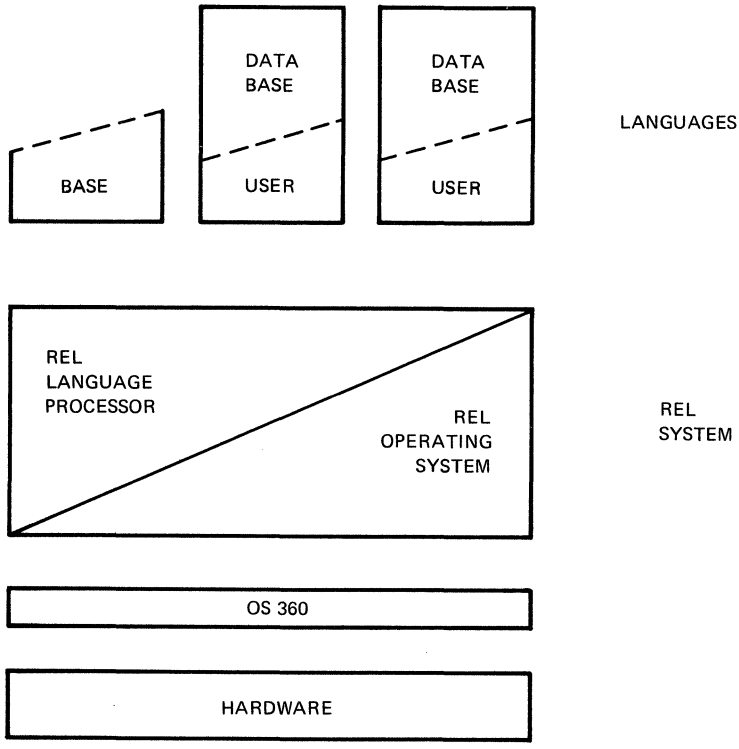


Fig. 19-1. REL Architecture

2. the language processor, which analyzes the incoming query or data and schedules and executes the appropriate calculations and processing of the data base;

3. REL languages and user language/data base packages.

One of the most distinguishing features in the architecture of REL as compared with other relational data systems is that it has a single language processor for all languages, and that this language processor is tightly coupled with the operating system. In most computing language systems, the system can accept and process statements of a given language by using a separate compiler specific to the particular language. REL, however, has a single language processor that can handle a wide variety of high-level languages. In essence, this language processor is a straightforward syntax directed interpreter. It includes built-in facilities for handling variables and recursion, and provides for extensions by users of the languages.

This feature of REL architecture has several major advantages. First, it is much easier to implement a new language or extend an existing one. Languages can be conveniently tailored to particular applications and specialized processes can be added to one's language as the need arises.

Second, since this single language processor is closely tied to the underlying operating system, it allows efficient scheduling, allocation, and access of peripheral storage, which could not be achieved otherwise.

Third, in applications where a number of offices or groups have their own "system," that is a language/data base package, the specific architecture of REL facilitates intercommunication between such subsystems.

The technical problems of implementing a natural language question-answering system are quite different from those encountered in programming language compilers. From the system programmer's point of view the relevant characteristic of most REL applications is that they deal with large data bases that must be kept in disk memory. The prime problem is efficient access to that memory. One solution is to restrict the nature of the questions that can be asked and optimize disk access methods around these restricted queries. This solution is not acceptable in the majority of applications to be found in dynamic situations. The REL solution, and a principal element of the system, are the paging algorithms for the dynamic optimization of access to the disk memory in terms of the data requirements of each query.

The language writer controls both the allocation of data to individual pages and the page segmentation of the interpretive routines, and he can do this without becoming involved in the details of the language processor or the paging mechanism. As a result, there is a rational relationship between lexicon and syntax, on the one hand, and the allocation and retrieval of pages from disk storage, on the other. Scattering of data and routines haphazardly over the peripheral storage, a source of major inefficiencies in other systems, is avoided.

One other design feature must be mentioned in connection with processing of data with complex interrelationships, to which experience with the system in Summer, 1970, pointed. An investigator who has a complex data base is soon led to ask questions that call for an extensive amount of computation and data manipulation. Such an investigator is usually well aware that he must wait a considerable time for his answer, and since he is aware of the amount of computation he has asked for, he is prepared for the delay in the response. This use pattern is likely to be quite typical, in a system where the user's language can be so easily extended, thus providing the means of succinctly expressing complex questions.

To facilitate this pattern of usage, REL will have the capability to cast off a query into the "background" as a low priority

job in the system and free the terminal for continued conversational use in the interim period. Since the REL operating system is a multiprogrammed system in which several jobs are resident in core memory at the same time, each occupying one of the available "slots," the plan is to make one dynamically allocated slot available for background jobs. Thus, one could say that the system would be both an interactive system and a conversationally driven batch system.

Such details of implementation have to be mentioned in view of our over-all objectives.

REL languages are of two types, which we call "base" languages and "user" languages. A highly specialized "user" language can be developed for a particular user, incorporating the syntax and basic algorithms natural to his problem area. More commonly, however, a user will make use of a general language already available, tailoring it to his own needs by introducing his own vocabulary and definitions.

Two such "base" languages have been implemented and applied by users. One is REL English and the other the REL Animated Film Language.

REL English, further discussed and illustrated in subsequent paragraphs, is a sizable subset of natural English. In the base version, the vocabulary is limited to the "little" words such as all, and, what, before. Together with the grammar rules for natural English, this constitutes a base on which a user can build his own special language, and then extend it and modify it according to his needs. As he makes use of the inherent definitional capability of the system, his language and his data base become tightly interwoven, constituting his own language/data base package. Several actual examples from user experience with REL English and the construction of a specialized language/data base package are given in Section III.

How, then, what exactly is meant when we refer to the "rapid extensibility" of REL languages? Our notion of extensibility derives from our understanding of how a researcher or any person dealing with dynamic data goes about his work. As his understanding of his material grows, he develops new concepts, finds new patterns in his data, interrelates his data in new ways. This evolving conceptualization is mirrored in his use of language. He defines new patterns and relationships in terms of old, and adds terms as he needs them. As he moves forward, he makes use of those newly defined terms and concepts. In dealing with his data, he needs to be able to communicate with the computer in these new terms rather than always having to express himself in some rudimentary language. Only in this way can be use the computer as a facile tool of his analysis.

The REL definitional capability can best be illustrated through the experience of two users who worked with those two entirely different languages: REL Animated Film Language and REL English.

Mr. John Whitney, the computer artist of international reknown, used the REL. AFL language. This language is a highly specialized language for conversational interaction with the graphic display terminal (IBM 2250) for composition and subsequent animation of motion picture films. Mr. Whitney used this language to make his film called MATRIX, which he presented at the International IFIP 71 Congress in Ljubljana, Yugoslavia, last August.[3]

In a typical working session, Whitney could define several visual forms, say a cube and a pleasing curve space. The definition of these forms might be either in terms of an array of simpler forms, e.g., line segments arranged to define the planar projection of the cube, or might involve mathematical expressions, for instance in defining space curve. Once defined, these forms could then be manipulated by the artist as conceptual units and be composed into higher level forms and sequences. For example, a series of cubes might move rhythmically along the space curve in such a way as to move into and out of symmetric interrelationships. The artist would then proceed by executing and modifying his developing composition on the display scope, working with the visual images to bring the ultimate composition into artistic balance. If it had been necessary for him to state these high level compositions in terms of the basic shapes of two-dimensional lines and mathematical equations, rather than in terms of cubes and the space curve, the artist would have been strained beyond his ability to conceptualize.

As far as REL English is concerned, this passage to new, high level conceptual forms can be seen in the protocol of Dr. Thayer Scudder who made extensive use of the experimental REL system. Dr. Scudder, a Caltech anthropologist, and Dr. Elizabeth Colson, of the University of California at Berkeley, used the REL system to analyze their data concerning the Gwenbe Tonga, a people living in Zambia. Their data base was of the order of 10^5 items. The following illustration is from one of Dr. Scudder's sessions with the computer. First he defined the term "sex ratio." Later on, he was interested in considering only the older women of the Mazulu village, whom he defined as "Mazulu crones." He could then ask:

"What is the sex ratio of the children of Mazulu crones?"

"What is the number of male children of Mazulu dames who were born before 1920, times 100, divided by the number of female children of Mazulu dames who were born before 1920?"

On the surface, this seems a minor advantage. However, in the process of ongoing investigation, the recogniting, testing and establishment of new conceptual forms is expected to take this step-by-step path.

These steps, as they build up, evolve into new and more revealing conceptual patterns. How a user extends his language through definitions during his ongoing conversation with the computer is well illustrated through Scudder's protocol. Definitions

can, of course, be deleted and changed, as well as added. The concepts defined, as well as the questions that can be asked, may involve higher level abstractions and complex interrelationships, not just simple identifiers of individual entities or subsets of the data, as might seem initially.

As a user builds up a hierarchy of definitions, computing efficiency is likely to be degraded when the higher forms are used, especially when they entail complex calculations on the data. The investigator should then have recourse to a programming staff who can replace the hierarchy of definitions leading to a term by an efficient algorithm expressing internally the complex meaning of the term.

Thus, REL provides for two kinds of language extension. First, it is easy for the investigator himself to define new terms and extend and modify his language, i.e., his lexicon and his data, while working with the data. Second, it is easy, at the programming staff level, to initiate and extend languages tailored to the needs of the users. It is precisely these two capabilities that constitute the extensibility provided by REL.

Finally, the data itself may need frequent extensions. There are two sides to this issue: adding small amounts of data, which the investigator can add just as easily as he adds definitions. Such additions are immediately incorporated into the data. The other side is adding large bodies of data, particularly when that data is on punch cards in typical field formatted form. Let us consider, for example, a data deck whose card format is:

NAME	POPULATION	LAND AREA
France	45540	213

Using a language based on REL English, an investigator could enter from a terminal the following definition:

```
def: "France" "45540" "213":

The population of "France" is "45540" and the land area of
"France" is "213".
```

The quotes indicate that any other similar term may be used in place of the ones shown, e.g., names of countries and other numerical data. It is easily noticed that this simple definition decodes the card format into a statement whose processing will build the facts indicated into the data base. Having submitted the card deck to the machine operator, one types: "Alternate input: cards." and the system then processes the data cards, whose translation is understood by the language processor in terms of the above definition.

REL English

REL English is currently the most prominent language within the REL system; it has already been tested extensively in user applications and has a variety of applications as a natural means of communication with the computer.

Just as in ordinary English we use different modes of expression, different styles to suit specific situations, in REL English not all constructions of ordinary English are available. For instance, colloquial, casual on the one hand, and extremely elaborate constructions on the other, are not part of REL English. However, we are continually bringing it closer to normal English by incorporating new structural features. Currently, REL English grammar consists of more than 350 rules which allow a variety of constructions to be handled. The grammatical structure of REL English is discussed in (4, 5, 6); here the presentation is limited to illustration of the constructions that can be handled and samples of actual conversations with the data.

As for the range of constructions handled, REL English uses:

1. Complex verb structures, including references to time; e.g.,

   Had John been given the message before his Boston friend arrived?

   Did John arrive in New York after July 1, 1970?

2. Relative clauses; e.g.,

   Did some boy see the girl who left London?

   Did John give Mary books which he bought from Tom?

3. Complex noun phrases; e.g.,

   Mary is the daughter of John's wife's brother.

   John sent a letter to his wife's mother.

4. Qualifiers, which select data and group it; e.g.,

   Which ships left Boston after May 1971?

   How many reports were sent by John last year?

5. Conjunctions, which join nouns and sentences; e.g.,

Did John live in New York or Boston?

Mary attended Harvard and her brother enrolled in Yale.

Combinations of such constructions, with some others not illustrated here, make it possible to use REL English with ease and a feeling of conversation in natural English.

To quote Dr. Scudder: "A great strength of REL is that the investigator can afford, in playing with the data, to search out a multiplicity of relationships, whereas in using other techniques he might settle on a single suspected relationship and after lengthy statistical analysis be tempted to read too much into correlations found."[7]

Comments such as Dr. Scudder's make those of us in the design and implementation of the REL system and REL English feel that we have already had very valuable experience with the system and we are confident of the REL promise.

With the tremendous developments in computing which we have witnessed in the past two decades, it is now time that computers should be "humanized" and that many men and women be liberated from the distance between men and machines. Computers should be easily manipulable tools in the hands of those to whose work they could contribute immensely – members of dynamic, complex environments. REL is a computer system for these types of users.

REFERENCES

1. Thompson, F. B., Lockmann, P. C., Dostert, B. H., and Deverill, R. S., "REL: A Rapidly Extensible Language System," Proc. 24th National ACM Conference, August, 1969.

2. Lockemann, P. C., and Thompson, F. B., "A Rapidly Extensible Language System: The REL Language Processor," 1969 International Conference on Computational Linguistics, Stockholm, 1969.

3. Whitney, John H., "A Computer Art for the Video Picture Wall," 1971 Congress of the International Federation of Inform. Proc. Soc., Ljubljana, Yugoslavia, August, 1971.

4. Dostert, B. H., and Thompson, F. B., "A Rapidly Extensible Language System: REL English." 1969 International Conference on Computational Linguistics, Stockholm, 1969.

5. Dostert, B. H., and Thompson, F. B., "How Features Resolve Syntactic Ambiguity," Proceedings, National

Symposium on Information Storage and Retrieval, University of Maryland, April 1971.

6. Dostert, B. H., and Thompson, F. B., "Syntactic Analysis in REL English," Proceedings, 1971 International Meeting on Computational Linguistics, Debrecen, Hungary, September 1971.

7. "REL Protocol: July 1970" California Institute of Technology, 1970.

20. A Computer-Directed Training System

John B. Goodenough

Softech, Inc.

This chapter will discuss one successful Air Force experience in using multi-access computers to help people learn to use a computer system effectively. It will then turn to some techniques which might be used in future systems.

Recent Air Force experience has centered on the Phase II Base-Level System, which consists of approximately 130 Burroughs 3500 computers located around the world. These computers calculate payrolls, assist in personnel management, and in general, accomplish functions which are common to Air Force bases everywhere. Some of the B3500 programs are interactive, in particular, the personnel management system; training personnel at each base to use this system effectively requires vast and continuing efforts. The training problem multiplies as more base level functions are supported by the Base-Level System.

In anticipation of these training problems, a system called the Computer Directed Training System (CDTS) was developed to train B3500 users on-the-job in the use of various functional programs supported by the B3500. (A course was also developed to train B3500 computer operators[1].) Within the last year, initial tests of one CDTS course have been completed, and the system is scheduled for world-wide implementation in January, 1972. Savings from the use of CDTS for this course alone are estimated at one million dollars per year, and moreover, the CDTS technique has achieved enthusiastic user acceptance.

CDTS is a simple system as computer-aided training systems go – it supports only simple training techniques. Specifically, it presents information or questions to a trainee, the trainee makes some kind of response, the system analyzes that response, presents either remedial material or new material, and waits for the trainee's next response. There's nothing particularly new about this style of instruction; it received the name "programmed instruction" years ago.

Certain management functions are also supported by CTDS; e.g., it keeps records of each trainee's progress and what kinds of errors he is making so that the course can be improved later.

The most significant aspect of the system is neither the method it uses to train people nor the training management functions that it supports, but that CDTS is available on the job. The console on which the trainee receives instruction is the same one that he will use to do his work when he finishes the course. CDTS is available on request and requires minimal trainee supervision even though it is a form of on-the-job training. The trainee's supervisor need not spend a lot of time making sure the trainee is getting the proper kind of training – the trainee just goes off and does it. Moreover, the trainee is on site while he is taking the course so that when a work crisis arises or questions arise in his area of responsibility, he is right on hand to help out. This is one reason why management likes an on-the-job training system. Another reason is that the personnel system course takes only 30 to 40 hours to complete, compared with three weeks of classroom training for the same course, and the computer-trained personnel perform as well or better than the classroom trained personnel. Finally, everyone in the personnel office can be trained to the same level of competence when CDTS is used, whereas at most 10 percent of the personnel are normally given the full classroom course.

Trainee acceptance has also been gained, despite a mean system response time of 44 seconds (with a large variance as well). All 25 of the original test trainees preferred this method of instruction to classroom instruction (although I should state that initially only 24 out of the 25 preferred the CDTS method; when the lone dissenter was asked why he preferred the classroom method he said he liked getting the per diem payments that came along with off-site classroom instruction. When we excluded this factor from his judgment, he admitted that he preferred the CDTS method).

From a cost viewpoint, direct costs of running CDTS come to about $2.70 per trainee hour as detailed below, as opposed to an estimated $5 per classroom hour.

Trainee management for CDTS costs cover management time required to identify the next person in the base personnel office who should receive training and similar functions. Classroom costs do not include TDY expenses of trainees, which can reach a considerable sum. Neither set of costs includes course preparation costs, costs for facility overhead or for course development. It is reasonable to ignore facility overhead costs, because the facility is required whether the training system exists or not. Course development costs come to between $70, 000 and $100, 000 per course. With respect to the CDTS personnel management course, course development costs are negligible, because, considering the number of trainees who should take the course

COMPUTING THE NUMBER OF PERMUTATIONS

INTRODUCTION

THERE ARE ONLY TWENTY-FOUR PERMUTATIONS OF FOUR OBJECTS, BUT THERE ARE 120 PERMUTATIONS OF FIVE OBJECTS AND 720 PERMUTATIONS OF SIX OBJECTS. IT IS OBVIOUS THAT AFTER A CERTAIN POINT THE LISTING OF PERMUTATIONS OF OBJECTS BECOMES AN ENORMOUS TASK.

DEFINITION

THE TOTAL NUMBER OF PERMUTATIONS OF A GROUP OF OBJECTS, OR ANY SUBGROUP, IS EQUAL TO

$$\frac{n!}{(n-r)!}$$

WHERE n = THE TOTAL NUMBER OF OBJECTS
AND r = THE NUMBER OF OBJECTS IN EACH PERMUTATION, SUCH THAT $0 \leq r \leq n$.

NOTATION

THE TOTAL NUMBER OF PERMUTATIONS OF A GROUP OF OBJECTS, OR ANY SUBGROUP, IS DENOTED BY THE SYMBOL

$$_nP_r$$

WHICH IS READ "THE PERMUTATIONS OF A GROUP OF n OBJECTS TAKEN r AT A TIME."

FORMULA

THE FORMULA FOR FINDING THE TOTAL NUMBER OF PERMUTATIONS, THEREFORE, IS

$$_nP_r = \frac{n!}{(n-r)!}$$

THE PERMUTATIONS OF n OBJECTS TAKEN r AT A TIME IS EQUAL TO . . .

THE FACTORIAL OF n . . .

DIVIDED BY THE FACTORIAL OF THE QUANTITY n MINUS r.

EXAMPLE ONE

AN EXPERIMENTER HAS FIVE VARIABLES WHICH HE WOULD LIKE TO TEST IN PAIRS. HE IS ALSO INTERESTED IN THE ORDER OF THOSE PAIRS. IF HE WANTS TO RUN ALL THE POSSIBLE TESTS ON THESE FIVE VARIABLES TAKEN TWO AT A TIME IN DEFINITE ORDERS, HOW MANY TIMES MUST HE RUN THE TEST?

$$_5P_2 = \frac{n!}{(n-r)!} = \frac{5!}{(5-2)!}$$

$$= \frac{5!}{3!}$$

$$= \frac{120}{6} = 20$$

Fig. 20-1. An Example of an Information Map.

each year (approximately 12, 000, due to the great personnel turnover experienced), the savings based on the cost analysis in Table 1 will be approximately one million dollars per year, exclusive of savings in TDY costs. Savings of this magnitude quickly amortize course development costs.

A few problems exist. First, the testing phase revealed that each base installation has too few consoles to train people on the job. The use of consoles for training had not been taken into account when the system was initially sized, and so while installations reported that they liked the training system, their workload was such that no time was available to use the consoles for training. The solution, of course, is to provide more consoles. The Air Force has estimated that 245 additional consoles will be needed for effective use of CDTS world-wide. Of course, these consoles need not be used solely for training – during peak workloads, they can be used for real work. Moreover, when a new system is made operational at a given site, the Air Force will import additional temporary consoles to handle the initial peak training requirement. The cost of the additional permanent and temporary consoles can be recovered easily from savings due to the use of the system. (It should be noted that as additional consoles are justified on a cost effective basis due to training needs, we come closer to the situation desired by the speakers at this meeting; namely, one console per person for his full-time use, be it training or work.)

A second problem is that the B3500 is heavily loaded. The large amounts of text required for the training system and its courses stretch online disk storage capacity to its breaking point. CDTS cannot be kept online at all times because of limited disk capacity, and when training system is in use, access to some other systems must be denied. Provision of more disk space is economically feasible, however, because of the magnitude of savings that can be realized from the full use of CDTS.

These two problems occurred largely because CDTS was an afterthought to the over-all system design. In sizing the system, CDTS requirements were not a factor. The lesson here is obvious – since simple online training systems are technically and economically feasible today, at least in the military environment, their demands should be considered from the start in system design and sizing. In fact, one of the main points I wish to make today is that training systems should be routinely provided (or at least, their provision should be explicitly considered) in future military multi-access systems.

One advantage of CDTS has still not been mentioned – ease of update. The procedures for using computer systems such as the personnel management system are constantly changing. Training materials must be revised accordingly. When training materials are published in book form, keeping the training manuals in agreement with the actual system is almost impossible. But with training

COST/CONSOLE HR.	
DISK STORAGE	$0.089
TERMINAL	$1.23
CPU TIME	$1.10
TRAINEE MANAGEMENT	$0.272
	$2.69
COST/CLASSROOM HR.	$5.00

Fig. 20-2. Costs per Trainee Hour

handled by CDTS, as the personnel system changes, corresponding changes will be made to the lessons. When the changed system is released, the changed training course will be released simultaneously. Thus all installations – world-wide – will have an up-to-date set of training materials, and the distribution of the changes is handled routinely – as part of its regular maintenance updates. If we were limited to printed training media – a book or something similar which, offhand, might seem to be a cheaper way of doing business – we would lose this simple method of keeping course materials up to date. This ease of change is really a tremendous benefit.

The CDTS system supports very well the teaching of a cookbook approach to problem solving. In the Personnel Management System course, a user is trained to process certain kinds of transactions using certain techniques available in the personnel management system. He is not trained to use a powerful set of data base update and query operations per se. To some extent, this is an advantage, because if a user has not dealt with a certain kind of transaction for some time, he can ask to be taken through the lesson module dealing with that sort of transaction again, i.e., the system can be used as a review mechanism by an already

trained user, even though CDTS is not particularly organized to facilitate review. And it is not set up to give a quick answer to a specific question about how to accomplish a certain action. Some technique other than the "programmed instruction" technique of CDTS is needed to meet the multiple requirements of users effectively, so that the computer can become more than just a training aid; it can become a continuing on-the-job performance aid as well, being continually available to assist the user in making effective use of his system.

One promising technique which overcomes the limitations of traditional computer aided training methods is information mapping[2, 3]. Information mapping was originally developed with the printed page in mind. One characteristic of information maps is the marginal notations which indicate in a general way the nature of each box's content. From one page to the next, the marginal notations are generally the same. The content in each box is grammatically separated from the content of adjacent boxes, i.e., there are no pronouns that refer back to a sentence in a preceding box. This means that when a person reads a printed page in this format, the order in which he reads the material is up to him. (Some people, for example, may prefer to read a definition, then read an example, and then look at the notation. Others may prefer to look at examples first, then the definition, and finally the notation.) With information mapped material, the reader can jump around quite readily, according to his own personal style of reading and learning.

The same information mapped page is suitable for initial learning and for review. For example, to review, a user can flip from page to page and look just at definitions and notation. The material is indexed by the marginal notations, and so is susceptible to use for different purposes.

Information structured in this way is particularly suitable for use in computer-based training systems. Each of the boxes on the printed page is a chunk of information of a particular kind, i.e., introductory, definitional, notational, etc. Given material organized in this fashion and a description of a user need, the computer can reorder the material to suit the particular need. For example, if a user requests a review of the material in the probability course, the computer system applies a system (or user) defined specification of what material is most appropriate for review purposes (e.g., just definitions, notations, and a single example), and then presents this material to the user. If he wants to reference a specific topic, he can enter a query and focus on a map or set of maps containing the information he wants.

So the point of information maps is that the same material is suitable for initial training, suitable for reviewing, suitable for referencing, and useful for just browsing through a course. A single preparation of the material can meet these diverse needs.

No computer-based system exists at present which uses information maps, although efforts are underway at the University of Pittsburgh and at The MITRE Corporation to develop such systems. A system design has been completed[4] using a set of decision tables which specifies how to sequence through material in various modes, but since these procedures have not been tested, undoubtedly some revisions will be necessary. It does seem likely, however, that a system which uses information maps will be more than just a training aid – it will be a job performance aid as well because of its utility for reference and review purposes.

CDTS and information mapped systems have a common characteristic, namely that all material presented to a trainee has been prepared in advance by the course author – the systems differ primarily in the extent to which the course material may be selected and rearranged by the trainee. In contrast, generative training systems use artificial intelligence techniques to create highly individualized questions and answers for a particular trainee. A course author in such a system does not prepare chunks of training material in advance, but rather develops a network of concepts and information about some topic. The system, using general rules, analyzes the network and either responds to questions by trainees or generates its own sequence of questions. Such a system is, potentially, capable of conducting a true dialogue between man and machine. Initial experimental results are promising[5]. At present, an experimental system containing information needed by ARPA network users is under design. The system will respond to queries in a seemingly intelligent fashion. For example, if someone asks how to use the Lincoln Laboratory computer, the system might reply that there are two computers at Lincoln Laboratory available on the network: an IBM 360 and the TX2. It would then ask which computer the user was specifically interested in. In another mode of use, the system would test a user's knowledge in a particular area selected by the user, and would proceed to fill in gaps revealed by the user's answers.

Of course, one realizes how speculative this sort of system is, but some success has been achieved in this direction and this is the kind of system that may well be in productive use in 15 or 20 years.

The three types of training and performance-aiding systems that I have discussed are distinguished in part according to the extent to which the computer or the trainee/user is in control of the dialogue. In the case of CDTS, the trainee has only very limited control over the system. In an information map system, the locus of control may vary more widely, although the style of the dialogue is still highly restricted. Finally, with generative systems, the locus of control may easily switch back and forth between user and computer, and the style of dialogue may appear quite free and natural. The Computer-Directed Training System,

information mapped systems, and generative systems are thus examples drawn from a spectrum of training and performance aids.

In conclusion, computer-based training aids can significantly help meet the training problems posed by sophisticated or even seemingly simple interactive, applications-oriented systems. According to Professor Corbató, the real system is the system that the user knows about. The only way he is going to get to know about the system is if he is provided with suitable training. And increasingly, at least in military systems, a computer-based training subsystem will be the most economical and effective means of providing that training. Unfortunately, neither users nor developers have the proper set of expectations about what training and performance aids a system should and can supply. No checklist for system design includes a training subsystem as a design possibility. There are systems where it is not worthwhile to implement computer-based training aids, but the decision not to use these techniques should be made consciously rather than by default.

Finally, computer-based training aids should be implemented in many systems and only when this is more widely realized and more systems have these training subsystems built in will the benefits of multi-access computing be fully realizable.

ACKNOWLEDGMENT

The work described here is and has been mainly the responsibility of Dr. Sylvia Mayer of AF Electronic Systems Division. Dr. Mayer's work over the past years and the many discussions we have had concerning the significance of these systems and concepts for improving the effectiveness of multi-access systems have been invaluable.

REFERENCES

1. ----, The development of a computer-directed training subsystem and computer operator training material for the Air Force Phase II Base Level System, ESD-TR-70-27 (AD 702 529), November 1969.

2. Horn, R. E., Nicol, E. H., Roman, R. A., et al. Information mapping for computer-based learning and reference, ESD-TR-71-165 (AD 729 895), March 1971.

3. Horn, R. E., Nicol, E. H., and Kleinman, J. C., et al. Information mapping for learning and reference, ESD-TR-69-296 (AD 699 201), August 1969.

4. Horn, R. E., Nicol, E. H., Roman, R. A., and Razar, M. E., Description of a computer-based learning-reference system for use with information-mapped data bases, Project Document No. 1, Information Resources, Inc., Cambridge, Mass., March 1971.

5. Carbonnell, J. R. and Collins, A. M., Mixed-initiative systems for training and decision-aid applications, ESD-TR-70-373 (AD 718 977), November 1970.

21. Integrative Analysis in Biology

Wilfrid J. Dixon
UCLA
Los Angeles, California

In this chapter, I wish to describe the Health Sciences Computing Facility and what it is attempting to do. I believe our approach to the pr oblem is different from those described in earlier chapters. In some sense, this chapter will discuss earlier chapters as well.

Our facility is located in the Medical Center at UCLA. It is supported by the National Institutes of Health to provide mathematical, statistical, and computer support for medical research. A large part of that activity is the continuing development of the BMD programs. Originally, these statistically-oriented programs operated in a batch environment. They have been widely distributed and provide a basis for many types of research. A more recent project is the support of research, development, and improvement of modeling systems. Major emphasis is now being placed on the development of interactive programs, some of them using graphical devices. As research tools, we have used the IBM 2250 and a variety of alphanumeric terminals. Interactive support brings a greatly increased power to the researcher. In addition to getting fast turnaround, he can interact directly with his analysis, and, with a graphical terminal, he has a pictorial output capability at his command. Programs are written not only by our own staff programmers but also by graduate students and various investigators who use our statistics and aid us in developing the computer support which makes these most effective. In addition to the systems to facilitate use and the package programs to provide the analytic tools, this has led us to the development of general purpose programs to aid the user in the development of special purpose tools for his needs. A retrieval program designed for handling tree-structured files is the first set of such programs planned for assisting users in the retrieval of data. Another application, which may be of interest

to some of the more system-inclined people, is a computer-aided instruction system. Using PL/1, we have written a set of macros which comprise a very effective means of developing teaching programs. Courses developed in this way are now in use for teaching the medical and dental students. The instructor merely specifies his statements and branching conditions and the macros are used to implement his requirements. The resulting systems run on an interactive terminal used by the student.

Effective tutorial and collaborative consulting is an essential part of our service to a user community which is predominantly inexpert in both computer usage and analytic techniques. Our plans for exporting our research and improving our local effectiveness entail expansion of our consulting capabilities. To this end, we are developing the capacity to permit two or more terminals to consult online, viewing the same output and responses, and commenting as required. This capacity is based on our "Interactive Consulting Program" – a set of modules which can be link-edited with any FORTRAN program to provide this capacity. ICP provides the system interfaces which enable the users to establish communication, view the same input and output, and interject comments. The first version became available in January, 1971, and we are finding it an effective way of carrying on communication at places both remote and close at hand. I want to stress here that two or more individuals are operating out of the same applications program at the same time in a facility. To facilitate program development by these users, we have concentrated on languages and systems which can be readily used for applications in medical research by people with little interest in or experience with computers. This has led us to the development of systems, special user-interfaces, and many applied programs. The graphical programs developed by our staff and our users have been written in either GRAF or PLOT, which are language extensions to FORTRAN and PL/1, respectively. These extensions are very simple to learn and use, so that anyone who knows either FORTRAN or PL/1 can become a graphics programmer in just a few hours.

The nongraphical interactive FORTRAN programs use a special version of the standard FORTRAN input/output routines. Input/output to the terminal is specified just like any other FORTRAN input/output. If desired, special control information may be specified in a COMMON block. Although it is usually necessary to re-work the control logic of a batch application program in order to take full advantage of the interactive capability, many batch programs become fairly satisfactory interactive programs when FORTRAN units 5 and 6 (usually the card reader and printer, respectively, in batch) are assigned to the terminal. Nongraphical interactive PL/1 programs use special input/output subroutines which are also quite easy to use.

All interactive jobs are handled on a time-shared basis, executing simultaneously with a full batch stream. This is carried

out under the control of our operating system, which we call TORTOS (Terminal Oriented Real Time Operating System). The system was developed jointly by IBM and members of our staff to meet the needs of this installation. TORTOS is based on the standard system 360 operating system with the MVT option, and consists of a set of added modules activated by a START command. Thus, as the additional releases of OS/360 become available we are able to add these modules to the standard operating system. Our operating system and interactive work were originally supported by an IBM System 360/75; in 1968, the system capacity was extended by replacing the central processor with a Model 91.

The basic structure of TORTOS permits almost total flexibility of use, but without special tools designed for user convenience its effective use requires considerable training. Now that our system is functioning well, we are turning our attention to expediting its use by beginners. The basic approach is being made with a terminal monitor. The monitor provides a simplified command language, with defaults and a prompting facility, which makes it possible to use it to solve relatively simple problems without first learning several different command language instructional sequences. It also makes it very easy for the user to perform certain very frequently used sequences of operations (e.g., compile, load and execute, look at termination codes, etc., edit programs, compile, – etc.). All monitor operations bypass the OS/360 job scheduling facilities, resulting in a very significant improvement in performance for the user.

Another significant development is TORTFORT, an interactive editor and FORTRAN compiler. With TORTFORT, the user may enter a FORTRAN program or specify a source program in our file service, and modify it, compiling the program line by line and correcting errors as they occur. When his entire program has been compiled and no errors found, he can execute it, still under the control of TORTFORT. Errors found during execution are indicated, and the user may return to the compiler or editor phase to correct his program. TORTFORT is easy to use and provides full facilities for the FORTRAN programmer. It is based on the University of Waterloo WATFOR compiler. The new version, based on WATFIV and offering greater convenience and efficiency, is near completion.

The guiding over-all impetus in the facility is that of serving medical research through mathematics and statistics, providing support with a computer operating in a time-sharing mode, and providing applications software that allows many problems to be solved with no changes to the applications program.

When I came into the biological area from the engineering research area, it took me some time to really find out that I was in a different culture. I think that that must be taken into account when one is thinking of computer systems and the man-machine interface. I feel that in some of the other chapters the man which

the speaker was interfacing with the computer was a man like himself. There are many other kinds of people in the world than those represented in this book. Biologists and medical people are really quite different. We have found this means we are now designing different systems than if we had tried in advance to guess what these researchers might need.

We find an integration of our techniques with our users' perspectives in almost any project we bring in. For example, the organ transplant problem is one of our important projects at present. The problem comes from the medical field, through a specialist in the medical area who knows something about the basic processes, the surgery, and so on. He comes to our facility to a mathematically or statistically inclined individual who then thinks about the role of the computer in the problem solution. So the interface has a long route to begin with, but, as the problem develops, even the most remote individual comes right up to the computer and starts interacting with it. This kind of a problem needs both new kinds of mathematics and new systems capabilities. We developed some new statistical techniques (e. g., one we call "Boolean factor analysis") which have been the basis for typing the white cells for organ transplant. We also found that the demands of this data system and the demands of the computations for the Boolean factor analysis really required new systems developments, so we have the man-to-man interface in developing the computer aspects.

Biological scientists stressed graphs before those in the computer field ever thought it might be a nice thing to do. You can see this in the journals of ten or twenty or thirty years ago. If one takes off the shelf a journal labeled "biology" and one labeled "engineering" or "physics, " one will find a wealth of graphic material in the biological field – ten to a hundred-fold more than in the others. So, in moving into graphics in this field, we found the people around us were really already graphics-minded, much more than those who were based in the computer science area. We have a very graphics-minded population with which to work, and they push us hard in the graphics area.

We have been trying for some time to find out why there isn't more interest in using color in graphics. In reading the literature on color, which exists in art, psychology, and the humanities, one will find they long ago suggested that mathematicians and engineers probably had a limited interest in color. However, since the engineer and the mathematician do not regularly read the biological journals, they are not impressed with the need for color and therefore may not be effective interfaces for bringing an additional dimension in the graphics field to the biologist or to others who think in color.

Perhaps some are in the computer field because a high school biology teacher tried to make them draw all of those graphics. The biology teacher was already involved in graphics

and has remained so. The mathematician and engineer are now ready to enter the field. Getting these people back together, I think, really must be done in the biologist's own environment. Perhaps I will be permitted to call that a graphic example of how one might well develop tools quite differently if the user population, which is really a different population, is steering and pushing the process which one is trying to develop.

PART IV. POLICY CONSIDERATIONS AND COMMENTARY

The two papers in this section were presented at the banquets and at the Commentary Session of the conference. They are quite philosophical in approach and, therefore, have been placed together.

Dr. Thompson presents a brief commentary on the character of ADP research today as reflected in the conference. He then discusses the need for a science of information and closes with some thoughts on the impact of computing on our society's organizational structures.

The last paper is a brief commentary by Dr. Syms on the conference. He closes with a discussion of the hypothesis that single-language/single-application computers may be so much more efficient than most current practices that we should consider them as the approach of the 70s.

22. The Need for a Science of Information

Fred Thompson

California Institute of Technology

Pasadena, California

I was asked to see if I could find some threads running through these chapters and to comment on them. But the one thing that stands out is their great heterogeneity and their incongruity. Each author reported from his own vantage point, seeing requirements and solutions and things to do from the point of view of his own experience in his own area, and these seldom matched the views of any other speaker. Many of the words used were at very high levels of abstraction, and as they got to that level, we could relate because we could also abstract our own experience to that level. This was markedly noticeable in the enormous number of cliches used. Yet, when we got right down to details, there was essentially no transfer value. For example, Dr. Bergman's report on "Real-Time Systems Requirements" was very convincing yet had no impact upon me at all, although I was completely aware of the thesis being spelled out and completely agreed with the speaker. It's a thesis favoring modular hardware that I am very familiar with because our own electrical engineers preach the same line very convincingly. I have no possibility of using such circuits both because of funding problems and, more important, because of political problems. I simply must use the general purpose computer at my installation. I would even like to go mini. I could afford a mini. In fact, it would be a considerable savings. I could use my grant money much more efficiently on minis but, because of the political situation in our own installation, I use the general purpose computer.

In a short conversation, I spelled out the capabilities of an English language direct-access system to several people at this conference, saying, suppose I had a magnetic tape under my arm and I could give it to you, could you use it? It turned out I was talking to some people who get thousands of bits per second off radars and have that as their only problem. So there was an

incongruity and we need to examine the implications of that to see what it means for the kinds of operations we ourselves do, to see what it implies for our community.

I would like to examine two major reasons for this incongruity and then I would like to comment on its implications and on what long-range solutions may exist.

The first reason is, certainly, a lack of any science of information. I should like to divide this notion of a science of information into two parts: the science of computers, what you might call computer science, and the science of human information or human information processing in social organizations. In information, we are beginning to have the basic units and basic conceptual ingredients of such a science. Certainly the work of Godel, Turing, and others in that area has given us a very firm hold on the notions of computability. The work of the theoretical linguists in the last several years has augmented that hold with a knowledge of language and its relation to semantics. On the basis of these results, we are beginning to put together a science of computing. Indeed, many contributions are being made at the present time. However, it certainly has not jelled into a form where we can make cogent comment upon the kinds of systems we are developing, the kinds of directions we should take.

To return to decision making and the area of the social interrelation of information, the area of management, which was very clearly pointed out as a major shortcoming by Commander Knepell, we find essentially no coherent body of data or of theory at that level at all. At the present time, most social scientists are not well enough equipped in mathematics to handle the material. Nevertheless, a growing body of social scientists is beginning to develop a common theory based again on mathematics – the mathematics of statistical decision theory, the mathematics of game theory, the mathematics of information theory tied in with the theory of automata and recursive function theory. This is not a single mathematics yet. But if we examine the various concepts across this rather broad field, we find a beginning of a unification of mathematics in that area. So I would like to say that the basic beginning of a social science, a quantitative and empirical social science is being made.

That basic beginning as it is now established, however, has one very bad flaw: All the mathematics and conceptual development underlying so much in this area, statistical decision theory on the one hand and artificial intelligence and simulation of cognitive processes on the other, seems to depend on a basic assumption that the various alternative states of the world can be adequately described in a given language. Notice, for example, in statistical decision theory, it is assumed that the consequences and alternatives existent in a given situation can be prescribed beforehand so that, on the basis of subjective probability decision rules, one can decide among them. In game theory, one presumes

that the strategy space is available, that both players are playing the same game and have equal understanding of the problem, that what they lack is information about the plays the other is making and the problem again is to choose from a predetermined set of alternatives. Information theory presumes a sample space is given and again it is a matter of choosing among alternatives. Certainly, in the artificial intelligence area, it is assumed that the language of the computer is adequate to describe the state in which the artificial intelligence is to perform.

This basic notion is a very dominant notion among those who are dealing quantitatively with social information. Even though this is a small part of the community, it is a growing part and a very powerful part. Certainly a dominant notion of all of the theoreticians that are working in an area need not necessarily reflect the reality about which it is supposedly built. Now I say, "need not necessarily" because I do want to take somewhat of an objective position in this regard. I believe that it does not at all reflect, that it indeed ignores, the very essence of the human cognitive process which I would like to characterize in a slightly different vein in a minute. In any case, it is certainly open to controversy. Indeed, there are very deep results in the literature, namely, the undecideability results of Godel and Tarski and others which, when brought into the balance, weigh heavily against the thesis that the states of the universe that are relevant to the management process can be formulated in any formal language. Thus they can certainly not be simulated on a computer. The situation then is simply that if we can formulate a total theory of, say, how a human being thinks, in a language that can be precisely described (and obviously if it can't be done on a computer), then the theory must indeed account for that language and must be adequate to describe it. Therefore, one falls into all the underlying antinomies. Consequently, there are deep reasons to believe that the human brain is not a computer and cannot be described in any formal language. That would apply also to the social processes in which we are involved, which would suggest the basic creative processes that underlie the processes of management most certainly are processes that cannot be simulated or initiated on a computer.

Now, these arguments are presented not to convince, but simply to give evidence that there are other, contrary arguments to the underlying thesis that all of these systems and system requirements can be adequately defined, can be put down in some precise way. There are alternative views. These views are well established in the philosophical literature. They have been discussed for eons and, indeed, with some mathematical bases that derive from some of the deepest mathematical philosophies of our decade. Consequently, we are in an area where there is deep controversy, even about those basic theories on which all our quantitative theories of computer science and

management science and artificial intelligence are now being based.

One of the most striking things about almost every paper presented here was the tacit assumption that the alternative space underlying the decisions that are to be made could be stated succinctly in a clear language whether they are procurement decisions among different computing systems, whether they are decisions concerning the kinds of systems that should be available to our managers, or whether they arise from decision processes in military establishment. All of the papers seem to make that underlying tacit assumption. And I raise grave doubt whether that can be done. In any case, we do not have an over-all theory of information and management in the broader sense. I don't think there's any serious worker in the field who would claim we did and that lack is contributing very heavily to the enormous heterogeneity and incongruity of the papers here.

The second and, I believe, the major reason for the amount of incongruity is that our social organizations have now come out of kilter, are incommensurable with the kinds of tasks we have to do. One may recall that the old farmer was born into a task-oriented environment. His job was to run that whole farm. He had to learn how to weld, to plow, to take care of fields, and to take care of animals. He had a great variety of skills to learn, all aimed at one single task which would occupy him all the rest of his life. In the development of our major systems, particularly the very large military systems, the split in the life of the development of a system falls along people lines. No one goes through the entire cycle of dreaming up the system, developing the system, learning all the skills of programming, carrying on through to management, growing up with the system, staying with it for a lifetime like the old farmer stayed with his farm, ending up as a system operator responsible for system operation in his old age. It is a way that we may wish to turn to in developing and living with the system.

I was particularly interested in Dr. Corbató's description of the programmers working in Multics – that many of them are growing up liking to live with Multics, building a whole career around a single system, being in on the design, continuing the development, upgrading the system, and looking forward to being in that particular system an entire lifetime. It would be interesting to see if we couldn't do this in the military where we hatch our officers in our schools of higher education, train them in particular military tasks involving the hardware, go on to develop systems for those environments, then go on to being the commanding officers in environments that actually use such systems. This may give a system a great deal more integrity than the way we do it now. Now, in spelling out this alternative, I'm simply trying to draw attention to the fact that we're not doing it that way and the very fact that the system developers have essentially no

experience in the management function again gives rise to myths that are believed by those people who produce systems which simply have no integrity when they get into the operational environment.

Let me spell out one aspect of that which occurred to me from the experience I've had working with the military in my past. It is this question of large data bases of up to a million items of data. I'm quite aware that most military people now consider such a data base to be very small, as 10^7 and 10^9 items of information is the order of magnitude of information that is being talked about. However, every experience that I've had and every officer I've talked to who has had direct experience with such data bases, confirm that those data bases are pure, unadulterated garbage. There are error-filled data bases such as, for example, a logistics data base for the Marine Corps which is estimated as having 60 percent of its entries incorrect. At one time $2 million was spent on a crash project headed by a very senior colonel in the Marine Corps to try to upgrade that data base so it could be useful. At the end of two years, the project was abandoned; it was recognized that there was no hope that it could indeed be put into any shape accurate enough for any honest use.

Let me give you another example of the same type. Many millions of dollars are spent each year on the Movements Report Section at the Chief Naval Operations (CNO). This is a section that opens teletypes to all major Navy installations where all flag-ranked officers and all major ships must report their positions within one-half hour of where they are at all times. This is a very extensive system, a very high priority system, and it presumably keeps the positions of these officers and ships tabulated in the Pentagon for CNO staff usage in a very accurate way. OEG did a study some years ago and asked what is the average positional error in ships underway (that is, ships at sea) as tabulated in the Movements Report and found that the average error was eight-hundred miles. No doubt there have been improvements in this system. However, it is clear these systems are grossly inadequate.

We considered the problems of privacy and their effect on the individual. On the other hand, these major data bases are such that the statistical summaries being taken from them are grossly inadequate to estimate the parameters they presumably measure. This is well known in the economic sphere, where many economists are completely convinced that the Bureau of Labor Statistics – such as, for example, the levels of unemployment – simply do not reflect the facts of life of our economy in any way that is adequate for establishing economic policy. The simple fact is that data bases of the size being talked about are not viable entries. Now there are good information theoretic reasons for this to be true, having to do with the rates of change

of the underlying conceptual elements that go into these bases. Officers and systems analysts who have had direct experience with these bases are very aware of these shortcomings. Now what we're doing is training very bright young men who are coming out of our colleges in the computer science area, to be excellent system programmers with broad knowledge of automata theory and recursive function theory; to be highly skilled men, often able to do fantastic jobs in artificial intelligence; but having absolutely no management experience, and no experience with operations, no experience in guiding through a large project, no experience with large data bases. They then make the completely fallacious assumption that we can get accurate information into a system that will give the CNO access to the position of every ship and every flag-ranked officer in the Navy at all times. They talk about unbiased and true situations, when every officer or every manager knows that the one thing he wants of his staff is good, honest bias, and we can't get that out of our computers. Consequently, we are designing systems that, when they get into the field, do not meet the requirements of their operating commands, largely because of the kinds of incongruities that cut the development of the system at all of its stages, from men experienced in one area to men experienced in another area to men experienced in a third area. As a consequence, we are losing congruity across the total community that deals with the kinds of systems we are talking about.

What are the implications, both social and scientific of these inadequacies, in our understanding of these kinds of systems? I think we should be hardheaded about this. We shouldn't go off crying in a corner. We should face up to what the situation is and what we should be doing about it. Certainly the first thing we've got to do is to increase the plurality of the kinds of systems we are developing, the kinds of effort we are making and the kinds of programs we are supporting. This is a pluralistic environment in which no sharp decisions should be made at this time. We should avoid, at all costs, major development programs that presumably give some sort of coherency to the kinds of programs and kinds of systems we are trying to develop. Rather, when a man stands up and shows by his experience that he is able to do something, we should give him the opportunity to prove himself. We should encourage individual efforts across a wide spectrum and watch the results of those programs carefully, choose those that are highly successful and push on with them. We should see that men of various points of view have the opportunity to step forward and do their thing and do it well. We've heard a lot about a lot of systems here. Some of the people who have been producing various sound and able ideas over a long period of time, often with very little financial support, are now beginning to be recognized as competent in this area and are being put in charge of larger systems and given larger responsibilities. This is a

wonderful thing to see. And we have had several people of that kind reporting at this conference.

The second thing we must do is drive, drive, drive for early operational implementation of these systems. We simply must stop trying to make these huge conceptual systems of great complexity. We've got to come right down to trying to get these systems into operationally significant and valid places early. For that is the only way we are going to learn what the problems are. We are going to have to iterate them, just as Dr. Corbató maintains. We iterate and iterate and iterate in redesigning and reimplementing. But the first thing I think that all of our development people and the development offices that support our grants should insist upon is aiming at early operations so as to get operational experience.

A third area is certainly that we should take more statistics on the operations of these systems. I think that Dr. Boehm was absolutely on the right track. We must have adequate statistics coming out of the system on its operation in live environments and then we must look at those statistics.

It seems to me these are the three main things that those of us working on systems and working with systems should do. Now the heavy responsibility in all this is on the research and development grant offices. I must say I not only take off my hat to them but I get down on my knees and pray for them. They have an enormous amount of leverage in this area. Thank goodness we've had ONR – which I consider to be one of the best agencies in this regard – that has indeed sponsored a great many and a great variety of things all across our country.

I'm very much worried about the ARPA situation right now. They can exercise enormous leverage on the community by applying political pressure to get into the ARPA net. This new effort of putting a lot too much money, as far as I can see, into this speech recognition problem just simply dries up the money for pluralistic efforts and aims at a far too narrow and too specific a task at this time when we know so very little. I think the R&D people have got to keep much more flexible. They've got to keep their grants low, and they've got to keep looking at the individual research grant. Now in this regard, the large contractors like SDC and SRI are going to have large marketing staffs that are going to push hard; they're going to try to sell, sell, sell to get the big grants. This is fine. That's what they should do in our capitalistic economy. This puts enormous pressure and responsibility on our grant officers, who must insist upon getting right down to the individual research team and evaluating that team on the substantive basis of what that team is doing and must not be hoodwinked into making large grants to large organizations who then can spread it around among many mediocre and a few good people.

In the longer range, I think there is hope for a much broader and more humanistic theory of information processing. It's one

of today's most challenging problems. I think that in the long run we will find an increase in our theoretical capability that will allow us to be more integrated and selective in the kinds of development we go into. But that is a long way off.

I think that another aspect of the situation is that of lowering costs of computer systems. This will be a very great blessing because it will allow us a greater pluralistic situation. We can afford to do a lot of things when the costs go down. Certainly the major limitation for a lot of us is this fact of hardware costs. Quite frankly, if I don't raise $35,000 a year for computing alone, I have no computer. I simply have to go out and raise $35,000. Now to get $35,000 for computer time, I've got to multiply that by at least four to convince the funding agencies that I've got a big enough project to afford $35,000 worth of computer. It's a fact of life for me. Many of us can't do many of the things we want to do because the computer costs are high. So one of the blessings we can look to in the future is lower computer costs.

Finally, I think that a major area that needs some fundamental looking into in this country is the revamping of our social institutions and social practices to cope with the kinds of problems we're coming up with. We're coming up with problems both in large and complex systems and in systems that are relatively short lived and of unstable time periods. The kinds of corporate organizations, the kinds of development organizations that we now have – in which we have this stratification of effort among people with varying experience who cannot talk to one another and a system that goes floating up through this – may be the wrong kind of social organization to encourage. To get really complex systems into the field and operating properly, it might be very much better if we could turn that stratification on its side and have people who receive their training in a technology, carry that technology into being, apply that technology to social problems, and die happy as managers of that technology while other developments are coming on to replace them. It may be a different kind of social organization and one more amenable to the kinds of social problems we have. It is important to recognize that there are alternative social organizations. And this, it seems to me, is one of the primary problems of our time, because it may very well be that we have outgrown the social patterns we now have.

23. The Trend Toward One-Language Computers

Gordon H. Syms
Navy Post Graduate School
Monterey, California

I agree with Dr. Thompson that the level of the talks was usually general and quite vague. I don't want to imply that that is a unique criticism of this conference; it is a much more general criticism. I would say the same about every conference I have attended in the last five years. I ask the question, How many here found solutions to his particular problems? This is quite a useful criterion for the success of any conference. If each person found one technique that helped him with one of his problems, or changed the way that he attacked one problem, then the conference was a success. If most people found a solution to one of their problems, then the conference was a great success, despite the vague and general nature of the talks.

We discussed communicating across disciplines; what about the communication within our own discipline? What have we done to improve communications between computer scientists? If we could solve this problem, we would really be taking a big step forward. We discussed adequate means of communicating ideas within a small group. We certainly have not come close to effectively communicating among groups of a hundred, and communicating in groups of four or five thousand, as at the Fall Joint Computer Conference, is just about hopeless.

About 1900, a study group proved that the lecture system was a poor means of communicating ideas, and the results of a discussion were retained much longer. Yet for some reason the lecture system continues to survive. Despite all the times I have tried to change my classes by using discussions, seminars or problem sessions, I usually return to the lecture system. Certainly all conferences use the lecture system. The main reason for this is that in the lecture system the person who stands on the podium looks very good. He has prepared for many hours on the subject of his choice; he can brag about his accomplishments;

he can demonstrate his superior knowledge; and only rarely is anyone in the audience prepared to challenge his facts or conclusions.

I disagree with Dr. Thompson about the discussion he singled out as being the least useful because the subject has been discussed at conferences for the last ten years. It was the one I found the most useful. Dr. Bergman talked about systems with many processes in parallel as opposed to all processes going through one or a few single ports, such as a few general purpose computers. He convinced me that, for some applications, new technology has made special purpose computers economically feasible, despite the improved capacity and reduced costs of general purpose computers. The problem that he did not address is the design cost of such systems. If that method is to be adopted, each special purpose processor must be designed at a relatively low cost. If the design cost is small, we don't have to worry about flexibility because the hardware can be redesigned and modified as cheaply as the software for the general purpose computer. I am very interested in lowering these design costs.

A concept developed by Bell and Grason for Digital Equipment Corp. is based on what they called Register Transfer Modules (RTM) which reduces the design cost. They developed a simple technique for determining and specifying the interconnections between the RTM, and thus were able to design a special purpose computer easily and efficiently using these modules. Their technique involved using a special flowchart, which was similar to a FORTRAN flowchart, to specify the interconnection of the modules. The flowchart then completely specifies the design of the computer hardware. Although there were 20 modules in total, the basic ones were a simple control module that transfers control to the next module, a decision module that transfers control to one of two modules on a condition, a general purpose arithmetic unit, a bus communication module, and a memory module. Using these modules they designed a simple computer. In fact, Digital Equipment now markets a small computer called a PDP-16 which they design to specification using these modules. They also claim that using these modules they can design the equivalent of a PDP-8 in six man-hours. Now that's really fast. I think the person must know a lot about the modules, a lot about the PDP-8 and be a real expert before he can do that; but if we're talking about designing a small computer in even one week we're talking about a relatively easy task.

I was interested when I saw this technique because I thought it had several possibilities. I looked at these modules and wondered how one could simulate the hardware and try some sample implementations. I looked at how to build a FORTRAN machine using these modules, and simulated some of the most difficult portions that one would encounter in building a FORTRAN machine. I also looked at the triangulation problem in airborne interception

to determine how big a problem that would be. The simulations showed that these problems could be solved quite easily by interconnecting register transfer modules as specified by flowcharts. I am now convinced that if you can draw a standard flowchart specifying a computer program for a problem, I can convert it to a register transfer module flowchart that specifies the interconnections of these modules. Thus, this technique is much simpler than other methods for designing special purpose hardware. Furthermore, it makes feasible the design of such items as hardware master executives, or monitors.

I believe the Navy could use the register transfer module design technique in several areas, two of which are special applications in NTDS (Navy Tactical Data Systems) and in the new Advanced Avionics Digital Computer. I have made a proposal for further development in this area and so I should have some further information in six months to a year. If we could cut the design cost substantially, then it becomes feasible to go back to designing processes in parallel and we can get rid of these monstrous operating systems we are all fighting.

ONE-LANGUAGE COMPUTER

I believe there is a trend towards networks of one-language computers. There are many one-language computers operating, many with terminals. For example, I know of 75 BASIC terminals running on a GE 635 and 50 APL terminals running on an IBM 360 Model 67 and using only about 25 or 30 percent of its resources. As soon as one switches to the general purpose multiple-language terminals, 30 terminals will completely load the same 360/67. So what we need is several one-language computers connected together so that any one can be accessed from any terminal, and files can be transferred from one to another. As well as the common languages like FORTRAN and COBOL, what kind of languages should be available on these processors? The language should be easy to use, it should be powerful and it should run efficiently. Let's look at these.

1. Easy to use

   It should be easy enough to use so that the man off the street or the high-level manager will be able to use this language. I don't care what he does with that language, but he must be able to do something in the same language that his system programmers are using. If a manager feels comfortable using the language that his computer people are using, even if he's only doing simple problems, he will spread the use of computers throughout his company. I think that APL is one language which in fact can do this. It can be used as a

desk calculator to do very simple things and yet it is powerful enough to do complex calculations, too. That brings me to the second point.

2. Power of the language

The power of a language is defined here as the ability to express complex operations in very simple terms. A language that is not powerful enough is not a good general purpose language because someone will quickly run out of power in that language and switch to one that makes his programming easier. This language must also serve an intermediate group of people, the once-in-a-month programmers. If it's not easy to use and powerful enough for this type of programmer, he will not use it. I think APL in fact does come a long way in that respect. This person is different because most of his effort is used in learning the language, which he doesn't want to do anyway, and little effort used in solving a problem; I contrast this to the next type who, as a system programmer, spends only a small portion of his total time in learning the language. He will learn a difficult computer language in order to get the powerful language he needs. I have a testimonial from an EE person who says he used to do his problems by forming a concept, translating it into mathematical notation, then into flowcharts, then into a programming language. Now he says that, with about six months' experience with APL, he skips all intermediate steps and formulates his concept directly in APL. He uses the power of the computer to help improve his concepts where they are weak and verify the results on test cases. Afterwards he may have to convert the results to mathematic notation or flowcharts in order to show someone else. An example of the power of APL is found in the calculating of the length of a vector which requires only one symbol in APL, compared to several statements in other languages. The trouble with all of of us is that we grew up with FORTRAN and so we think FORTRAN (or COBOL), and it's going to be a long time before we really start thinking in parallel or matrix code. The worst thing is that if one starts thinking in that way he always gets clobbered by the amount of computer time he uses. This brings us to the third point.

3. Efficiency of implementation

The operation must be efficient. This brings us around to the special-purpose APL machine. We could use

> conventional designs and I know of two already, but that's a lot of work. Perhaps we could use microprogram machines, but I would like to propose another alternate, which is using the register transfer modules I spoke of earlier and interconnecting them. I have not attempted this problem but I am certain that the technology is at the point where we could build a special-purpose machine, and fairly easily. Anyone planning to crusade to get this language or equipment accepted has a lot of momentum to buck.

Now I'm not suggesting that APL is the answer to all the languages. It has some very major shortcomings. In most implementations, for example, it limits the size of the workspace (memory size of a program). That restriction could be easily changed. It also does not have decent I/O particularly with respect to disk files, card readers and line printers. Finally, one must be able to produce compatible files that can be used on other computers using other languages.

In conclusion, I would like to see somebody become interested in developing an APL machine. I'm not currently working in that area at all but I have some very strong ideas. Incidentally, the developers of the CDC Star have hired some people to look at implementing APL on that machine. That should be a powerful one.

Other books of interest...

PARALLEL PROCESSOR SYSTEMS, TECHNOLOGIES, AND APPLICATIONS

Edited by L. C. Hobbs and D. J. Theis, Hobbs Associates, Inc.; Joel Trimble, Office of Naval Research; Harold Titus, Naval Postgraduate School; Ivar Highberg, Naval Weapons Center.

Providing a much-needed exchange of information, this book brings together a collection of papers by active workers from system, device, software, and application disciplines. Directed primarily toward those involved in the design and utilization of parallel processor systems, it also provides newcomers to the field with an excellent overview and introduction to the subject. #9175, 448 pages, 6 x 9, cloth.

FILE STRUCTURES FOR ON-LINE SYSTEMS

By David Lefkovitz, Moore School of Electrical Engineering, University of Pennsylvania.

Widely adopted, this well-written, illustrated volume is an invaluable aid to the programmer, analyst, and student. It provides the programmer or systems analyst with basic principles, as well as specific techniques, for the organization of files in mass random access computer storage. Design principles are given which illustrate broad areas of application and cost/performance trade-offs. #5943-4, 232 pages, 6 x 9, cloth.

OPERATING SYSTEM ANALYSIS AND DESIGN

By Leo J. Cohen

A broadly based text that determines the fundamental properties of operating systems, and develops tools that are applicable to their formal design and analysis — particularly multiprogramming systems. The author orders and organizes the materials of a subject for which there has been no acceptable definition. #5643-5, 192 pages, 6 x 9, cloth.

HAYDEN BOOK COMPANY, INC.
Rochelle Park, New Jersey

ISBN 0-8104-5964-7