USC/INFORMATION SCIENCES INSTITUTE 4676 Admiralty Way Marina del Rey California 90291

ARPA ORDER NO. 2223

ISI/SR-77-8

1977 Annual Technical Report

A Research Program in Computer Technology

July 1976 - June 1977

Prepared for the

Defense Advanced Research Projects Agency

UNIVERSITY OF SOUTHERN CALIFORNIA

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (14) ISI/SR-77-8	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER (9)	
4. TITLE *(and Subtitle)* (6) 1977 Annual Technical Report: A Research Program in Computer Technology	5. TYPE OF REPORT & PERIOD COVERED Annual Technical Report, July 1976-June 1977
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) ISI Research Staff	8. CONTRACT OR GRANT NUMBER(s) (15) DAHC 15-72-C-0308; ARPA Order-2223
9. PERFORMING ORGANIZATION NAME AND ADDRESS USC/Information Sciences Institute 4676 Admiralty Way Marina del Rey, CA 90291	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS ARPA Order #2223 Program Code 3D30 & 3P10
11. CONTROLLING OFFICE NAME AND ADDRESS (11) Defense Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209	12. REPORT DATE September 1977
	13. NUMBER OF PAGES 108
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* ------- (12) 115 p.	15. SECURITY CLASS. *(of this report)* Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

This document is approved for public release and sale; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

----

18. SUPPLEMENTARY NOTES

----

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

1: abstract data type, abstraction and representation, Alphard, Euclid, interactive theorem proving, Pascal, program correctness, program verification, software specification, verification condition.

2: computer terminals, interactive message service, Military Message Experiment, nonprofessional computer users, SIGMA, terminal-based

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This report summarizes the research performed under Contract DAHC 15-72-C-0308 by USC/Information Sciences Institute from 1 July 1976 to 30 June 1977. The research is aimed at applying computer science and technology to areas of high DoD/military impact. The ISI program consists of eleven research areas: Program Verification - logical proof of program validity; Information Automation - development of a user-oriented message service for large-scale military requirements; Specification Acquisition From Experts -

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

401 ...

19. KEY WORDS (continued)

2: message service.
3: abstract programming, domain-independent interactive system, natural language, nonprocedural language, nonprofessional computer users, problem solving, problem specification, process information.
4: computer network, digital voice communication, network conferencing, packet-switched networks, secure voice transmission, signal processing, speech processing, vocoding.
5: command and control, computer graphics, high-level graphics language, on-line map display.
6: ARPANET, control memory, emulators, microprogramming, microprogramming language, MLP-900, National Software Works, operating systems, program development tools, resource sharing, TENEX, time sharing, writable control memory.
7: access control, computer security, encapsulation, error analysis, error-driven evaluation, error types, evaluation methods, protection mechanisms, writable control memory.
8: communication protocols, packet radio network, sensor clusters, sensor networks.
9: command and control, digital voice communication, graphic input device for terminal, portable terminal, radio-coupled terminal.
10: ARPANET, naive computer users, TENEX, user assistance, user documentation.
11: ARPANET interface, computer network, FPS AP-120B, KA/KI, KL1090T, KL 2040, PDP-11/40, PDP-11/45, resource allocation, TENEX, TOPS-20.

20. ABSTRACT (continued)

the study of acquiring and using program knowledge for making informal program specifications more precise; Network Secure Communication - work on low-bandwidth secure voice transmission using an asynchronous packet-switched network; Command and Control Graphics - development of a device-independent graphic system and graphics-oriented command and control applications programs; Programming Research Instrument - development of a major time-shared microprogramming facility; Protection Analysis - methods of assessing the viability of security mechanisms of operating systems; Distributed Sensors Network - developing algorithms and communication protocols to support the operation of geographically distributed sensors; Packet Radio Terminal Mockup - development of a mockup of a minimum-size, low-power display terminal to illuminate human factors issues in portable environments; User-Dedicated Resource - assistance to and documentation for ARPANET users worldwide; and ARPANET TENEX Service - operation of TENEX service and continuing development of advanced support equipment.

USC/INFORMATION SCIENCES INSTITUTE 4676 Admiralty Way Marina del Rey California 90291

ARPA ORDER NO. 2223

ISI/SR-77-8

1977 Annual Technical Report

A Research Program in Computer Technology

July 1976 - June 1977

Prepared for the

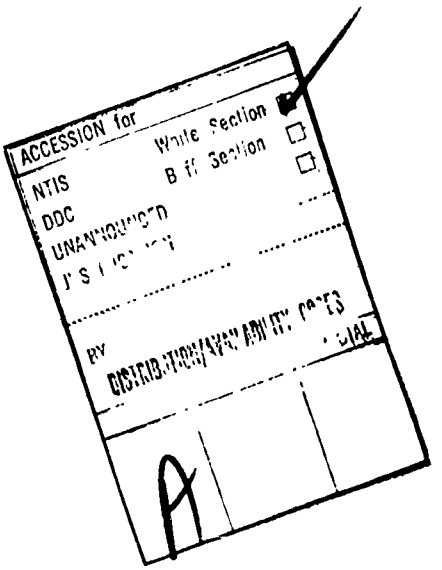
Defense Advanced Research Projects Agency

Effective date of contract:	**17 May 1972**
Contract expiration date:	**30 September 1978**
Amount of contract:	**$21,407,016**
Principal Investigator and Director:	**Keith W. Uncapher** **(213) 822-1511**
Deputy Director:	**Thomas O. Ellis** **(213) 822-1511**

This research is supported by the Defense Advanced Research Projects Agency under Contract NO DAHC15 72 C 0308, ARPA Order NO. 2223, Code NO. 3D30 and 3P10.

UNIVERSITY OF SOUTHERN CALIFORNIA

CONTENTS



SUMMARY

This report summarizes the research performed by USC/Information Sciences Institute from July 1, 1976 to June 30, 1977. The research is aimed at applying computer science and technology to areas of high DoD/military impact.

The ISI program consists of eleven research areas: *Program Verification* - logical proof of program validity; *Information Automation* - development of a user-oriented message service for large-scale military requirements; *Specification Acquisition From Experts* - the study of acquiring and using program knowledge for making informal program specifications more precise; *Network Secure Communication* - work on low-bandwidth, secure voice transmission using an asynchronous packet-switched network; *Command and Control Graphics* - development of a device-independent graphic system and graphics-oriented command and control applications programs; *Programming Research Instrument* - development of a major time-shared microprogramming facility; *Protection Analysis* - methods of assessing the viability of security mechanisms of operating systems; *Distributed Sensors Network* - developing algorithms and communication protocols to support the operation of geographically distributed sensors; *Packet Radio Terminal Mockup* - development of a mockup of a minimum-size, low-power display terminal to illuminate human factors issues in portable environments; *User-Dedicated Resource* - assistance to and documentation for ARPANET users worldwide; and *ARPANET TENEX Service* - operation of TENEX service and continuing development of advanced support equipment.

EXECUTIVE OVERVIEW

The Information Sciences Institute (ISI), a research unit of the University of Southern California, performs research in the field of computer and communications sciences with an emphasis on systems and applications.

The character and uniqueness of ISI are expressed in the following objectives:

- A major university-based computer science research center.
- A center which possesses a unique blend of basic research talent and application and system expertise. The last two attributes are of special significance to the application of computer science and technology to key military problems.
- A university-based research center with strong active ties to the U.S. military community and a strong leadership role in identifying key computer R&D requirements in support of long-term military needs and in assuring responsibility for solutions for some of the problems as well as an active, successful role in several key research areas.
- ISI also embraces a very significant support role as a major TENEX resource on the ARPANET.

A close relationship is maintained with USC academic programs through active cooperation among the Institute, the School of Engineering, the Department of Electrical Engineering, and the Computer Science Department. Ph.D. thesis supervision is an integral part of ISI programs, as is active participation of research assistants supporting ISI projects. ISI staff members frequently direct or participate in nationwide and international meetings and conferences; the Institute also hosts frequent colloquia and seminars as a forum for distinguished speakers from other organizations. ISI hosts a very large number of senior military officers as a means of providing insight to the potential of information processing in military mission accomplishment.

The activities of ISI's eleven major areas of research and associated support projects are summarized briefly below. Some of the research projects reported in this document are discrete activities in themselves; others can be seen as parts of a larger whole. For example, Program Verification, Specification Acquisition, and the Programming

Research Instrument projects should be considered as individual parts of an overall research effort in Programming Methodology; Information Automation, Network Secure Communication, Distributed Sensors Network, and Command and Control Graphics are linked elements of a major investigation into Network Communications Technology. These mutual interdependencies among the various projects at ISI contribute largely to the fruitfulness of the Institute's research activities.

Program Verification. The goal of program verification research at ISI is to develop an effective program verification system for proving that computer programs are consistent with precisely stated detailed specifications of what the programs are intended to do. The system is expected to replace significant parts of testing in current software development, and will also provide important tools for developing and judging the success of new programming language designs, new programming methodologies, and new detailed specification techniques. Already running at ISI is an initial, experimental version of an interactive program verification system whose design philosophy is to provide automatic assistance for the verification process where practical, and otherwise to rely on human interaction. The system has verified numerous example programs. Important progress has been made in the following areas: improved user environment and interface to the verifier, extensible verification generator, algebraic approach to data abstractions including their verification, and influence of verification on language design. The eventual impact will be an increase in the quality of software.

Information Automation. The Information Automation project has developed a secure, on-line interactive writer-to-reader message service for the military community. Such an on-line service, new to the military, provides interactive assistance for formal messages from the initial draft preparation through coordination, transmission and distribution. In addition, it provides informal secure "off-the record" communication to reduce the need for face-to-face meetings. In 1975 ARPA, the Navy, and CINCPAC agreed to an operational test of an experimental message service, to be run at CINCPAC staff headquarters, Camp Smith, Oahu. In February 1977, ISI's SIGMA system was selected in a competitive evaluation as the message service best suited for the experiment. The Military Message Experiment test, funded jointly by NAVELEX and DARPA, is designated to begin in January 1978 and to run for two years.

Specification Acquisition From Experts. The major effort of the SAFE project is simply to allow users who are not computer programmers to functionally specify their application directly to a computer system, with the system transforming this input into a precise functional specification of the application. This system is intended to be both independent of any particular problem domain and able to deal with "loose" (i.e., incomplete, inconsistent, etc.) problem-oriented descriptions of a domain through a dialogue with the user. From this dialogue the system can acquire the "physics" (the objects, laws, relationships, etc.) of the loosely-defined domain, structure it, and use it to understand further communication and finally to rewrite the specification in precise operational form. The system is being developed in the context of simplified real-world

military specification manuals. Several such simplified specifications have been successfully converted to a precise operational form. The system is now being expanded to deal with more complex specifications.

Network Secure Communication. The major objective of ARPA's Network Secure Communication project is to develop secure, high-quality, low-bandwidth, real-time, two-way digital voice communication over packet-switched computer communication networks. This kind of communication is a very high priority military goal for all levels of command and control activities. ISI's role in this effort is to develop efficient user-oriented systems for digital voice communications, primarily over packet-switched networks such as the ARPANET. The ISI NSC project is working on network voice protocols, digital voice conferencing systems, voice-oriented network host operating systems, real-time signal processing, hardware development, and future integration of voice, graphics, and text into a powerful packet-based command, control, and communications system. During the past year the ISI NSC project implemented and tested a very low bandwidth variable-rate Linear Predictive Coding vocoder, developed a file format and network protocol for storage, retrieval, and manipulation of voice files (and other related data), and implemented a greatly improved software and hardware environment for real-time packet speech research and development.

Command and Control Graphics. In cooperation with the Navy, ARPA/IPTO is preparing a command and control testbed at Naval Ocean Systems Center to explore and demonstrate the use of advanced computer techniques in military applications. There is a recognized need for high-quality graphics input and output in this environment to accommodate effective data presentation and to support high-quality user interaction with command and control programs and data bases. In support of the testbed, the ISI Command and Control Graphics project is developing a graphics system that homogeneously supports graphics terminals of widely varying capability, potentially interacting with one or more different programs running on separate computers connected via a command and control communications network. The project is also developing advanced applications for the testbed that fully exploit the graphics medium.

Programming Research Instrument. PRIM is an interactive microprogrammable environment used for the emulation of existing or newly specified computer systems with major emphasis on providing debugging tools. These tools, available via the National Software Works, provide the users with more powerful debugging facilities than available in the original target computer systems. The facility consists of a powerful microprogrammable computer (MLP-900), closely coupled to a TENEX operating system, and software to permit users to create and debug new emulators and target systems. Two prototype emulators, the UYK-20 and the U1050, have been completed and have been integrated into NSW. PRIM is a generalized solution to the problem of software development through the use of emulation tools.

Protection Analysis. The goal of this project is to develop effective techniques and tools for detecting in operating systems various types of protection errors, i.e., errors that allow the security of systems to be compromised. The approach is empirical, based on the observations that (1) protection errors fall into a limited number of distinct types and (2) techniques can be developed for finding the errors associated with each type. The method is to identify the error types and analyze each type to describe its characteristics for the purpose of developing an effective search strategy. To date, errors and corresponding search techniques of several types have been reported.

Distributed Sensors Network. The objective of this project is to investigate the impact of communication technology on detection technology, and to demonstrate that a distributed network of remote (and possibly different) sensors interconnected by apropriate communications is a cost-effective way to solve the detection problem. A model is used consisting of clusters of sensors, interconnected via a packet radio network, with each cluster controlled by its own central controlling logic. This lower level of the system is referred to as the "intracluster communication." The higher level of the model is a network of clusters and processors cooperating with each other, using external knowledge when necessary and feasible; this is referred to as the "intercluster communication." Protocols for both the intracluster and the intercluster communication are investigated.

Packet Radio Terminal Mockup. ISI designed and produced a model to illustrate the approximate size and weight of a minimum-size (maximum portability) radio-coupled terminal (outside dimensions 6-7/8" x 3-7/8" x 1-1/2"). The terminal was assumed to have the following characteristics: all normal alphanumeric computer terminal functions with full ASCII keyboard, display of 25-80 character lines, upper and lower case text, and inverse video highlighting; 480 x 250 raster point graphics or Fax; direct tablet over display with stylus graphic input; digital voice commmunications; integral packet radio transceiver; and self-contained rechargeable batteries for at least 10 hours operation.

User-Dedicated Resource. This project was begun in order to provide encouragement and help to new ARPANET users who faced an initial barrier because their lack of previous on-line experience, the rapidity of systems turnover, and the insufficiency of introductory documentation. New ARPANET users are contacted as soon as their accounts are installed on the ISI machines; they are interfaced to the available network facilities by means of three levels of appropriate documentation. Both short-term solutions (answering questions, solving individual problems) and long-term solutions (guiding users in solving common problems, making available better procedures and documentation) are routinely provided. Introductory primers and manuals have also been written exclusively for new users.

ARPANET TENEX Service. ISI is supporting, operating, and maintaining four complete TENEX systems at ISI on a schedule of 161 hours per week each, in order both to provide TENEX service to ARPA and to support its research projects via the facilities at ISI.

ISI also operates three TENEX systems at a computer center that is part of the command and control testbed at the Naval Ocean Systems Center, San Diego, California.

The Institute provides a 24-hour availability of TENEX systems, maintenance, and operators, continued development/improvement support, support of the XGP at IPTO, as well as ARPA NLS user support and minimal NLS software support. Through this support we have achieved increased long-term up-time, faster repair and improved preventive maintenance, economy of scale in operation, and the benefits of ISI expertise in establishing requirements for optimal loading and high reliability. In addition, this experience is used to assist in improving system reliability and to increase the number of users which can be handled with the required response time.

1. PROGRAM VERIFICATION

Research Staff:
Ralph L. London
Raymond L. Bates
Susan L. Gerhart
David R. Musser
David S. Wile
Martin D. Yonke

Research Assistants:
Donald S. Lynn
Mark S. Moriconi
David G. Taylor

Support Staff
Betty L. Randall

INTRODUCTION

In many areas the consequences of a program not performing as specified can be very severe in terms of cost or damage. Our ultimate research concern is to improve the quality of computer software by making it possible to demonstrate that computer programs meet detailed functional specifications. The demonstrations could take various forms--for example, testing, simulation, or code reading by humans--each of which is intended to (and does) increase one's confidence in a program. For us, however, program verification means providing a mathematical proof that a program is consistent with its specifications. Although no absolute guarantees can be provided by this or any other form of demonstration, the discipline of mathematical proof and the assumptions it identifies in the program combine to provide significantly increased assurances that a verified program and its specifications are consistent. Through the development of libraries of well-specified and verified programs, it should also be possible to reduce the overall cost of creating and maintaining software.

Verifications of programs have been produced by hand and by computer. Because neither approach strictly by itself can verify very many programs, we are producing verification tools in the form of computer programs to carry out automatically some of the necessary proof steps and to seek and accept human guidance in other parts of verification.

Increasingly, we have moved away from verifying arbitrary pieces of code, especially those to which little or no thought to verification was given during program construction. We have been involved in designing new programming languages, Alphard [Wulf76a,b, Shaw77, London76] and Euclid [Lampson77, Popek77], in which ease of verification of the resulting programs has been one of the goals of the language design from the start. The influences of verification concerns have been of considerable importance in the resulting designs as described elsewhere [London77a]. In brief, verification has suggested appropriate questions during the design which we feel certain

would have been overlooked on the basis of other concerns alone. Verification has also provided fairly objective ways to evaluate various decisions. Of specific note is the discipline of constructing axiomatic definitions, i.e., proof rules (rules of inference), for the languages as a very effective way to uncover consequences of various decisions. While it may appear that verification has the final veto power in the designs, the fact is that other considerations nearly always supported the verification suggestions. In the documents describing requirements for proposed common DoD High Order Languages [DoD77], one can see the concern for verification expressed; in particular, the role of proof rules in the design is recognized.

In these languages, an important methodological concern is the use of abstraction--retaining essential properties of objects while neglecting inessential details. The languages have features to encourage the use of abstraction, both data abstractions and control abstractions. While abstraction has numerous programming benefits, the major benefit for us is in the verification of the resulting programs. At the proper level the necessary details are verified once per definition of the abstraction rather than once per use of the abstraction. In turn, each use of the abstraction is verified by employing the result of the verification of the definition without reverifying that result [London76, Guttag76]. Structuring programs by using abstractions in these ways also has important consequences in the often neglected area of program modification. If changes in representations of abstractions can be made without changing the abstract properties (as is often the case), it is necessary to reverify only the definition and not each use of the abstraction. Indeed, we have verified modified programs by changing the proofs only in ways corresponding to the program modifications; it is unnecessary to reverify the entire program from the beginning. The real significance of this development is the ability to verify reasonably large programs by verifying in turn smaller, segregated units of the overall program and then using the results hierarchically in a way that reflects the hierarchy of the overall program.

Specifications of well-structured programs can be expressed in several complementary ways. We are exploring a method called algebraic axioms for specifying the abstract properties of data by means of equations relating the operations on that data. This approach permits a self-contained specification of a data type, independent of any particular representation of the data. A complementary approach which we are also studying is abstract model specifications: the use of a representation of the data in terms of some well-known mathematical objects such as sets, sequences, or tuples. We have also been concerned with specification of error handling, developing new techniques that permit precise and rigorously checkable specifications of behavior of programs under abnormal conditions.

VERIFICATION TOOLS

We have continued development of the XIVUS system for verifying properties of Pascal programs. The improvement of a number of components of the XIVUS system during the past year has included the development of a new parser and a new verification condition generator. The parser accepts the full Pascal language and a more flexible assertion language. Additionally, some language features of Euclid, such as imports lists, are permitted. Because the parsing mechanism is really a parser generator, other languages and incremental changes can easily be accommodated. An experimental version of an Alphard parser was implemented using this facility.

The new verification condition generator extends the subset of Pascal constructs allowed to include case statements, records, and arrays with more than one dimension; Euclid-like import lists are also processed. Parameters to procedure calls are permitted to be components of arrays or records (implementing the rules for Euclid parameter passing proposed in [Guttag 77]).

A new rule of inference for functions [Musser77b] was developed that permits a simpler method of generating verification conditions for Pascal functions, and this method was implemented in the verification condition generator. The rule treats functions as a separate programming construct rather than including it as part of rules such as assignment, conditional, iteration, and procedure. The rule also avoids unsoundness which can occur with certain published function rules when functions fail to terminate (such problems did not occur in the previous verification system which included functions as part of other rules). The new rule is especially appropriate for use with algebraic axiom specifications of functions.

Proof rules were also constructed for the Euclid language [London77b]. The rule for the Euclid module construct is new; the process of constructing it identified trouble spots with certain previous ideas of Euclid modules and also provided a measure of the complexity of the module concept. The procedure rule removes restrictions from other rules for procedures, properly covers the static scoping of "global" variables defined for Euclid, and depends crucially on the lack of aliasing of procedure parameters. Details of the Euclid procedure rule are described in a separate paper [Guttag77].

Using the XIVUS verification system, researchers at the UCLA Security Project have successfully verified several procedures in their security kernel. At their request, several new features were added to XIVUS. A serious security programming error was uncovered by the verification discipline, specifically the need to write specifications.

In order to explore the potential of algebraic axiom specifications, we have designed and implemented, in Interlisp, a "Data Type Verification System" (DTVS) [Guttag76, Musser77a]. DTVS interactively accepts specifications and implementations of data types and performs three classes of operations on them: (1) storage and retrieval for display or

manipulation; (2) testing of specifications using direct implementations and expression reduction systems [Guttag76]; (3) carrying out proofs about data types, particularly verifications that a particular implementation satisfies the specification (the equations) of a data type. All user input is subjected to strict type checking, using declarations of variables and interface specifications of operations. From axioms or other equations in the input, the pattern compiler produces a set of Interlisp functions which, when executed interpretively or as machine code after being compiled by the standard Interlisp compiler, have the effect of applying the original equations as rewrite rules.

The core of the proof system component of DTVS is a "conditional evaluator" called CEVAL which carries out proofs mainly as a series of reduction steps. A verification condition generator component uses the specifications and implementations of data types to set up sufficient conditions for correctness of the implementations. The proof system keeps track of the status of the proof of each of the verification conditions and any assumptions made (interactively) during the course of the proof, plus enough other information that the current stage of the proof can be recreated automatically by a "recheck" command. By making a transcript of the operation of the proof system (including, optionally, tracing of the application of each axiom or lemma) during a recheck of a completed proof, one obtains a detailed documentation of the proof which makes explicit the assumptions on which it depends.

Among the specification and proof techniques developed in the DTVS system is a new method of specifying preconditions and error checking and an associated proof method which permits a more extensive use of unconditional rewrite rules than previously possible. Consequently, proofs about programs that potentially generate errors can be carried through more automatically. These techniques have been used to specify and verify an implementation of a word-frequency table of bounded size.

A major concern of our investigation of algebraic axioms is combining algebraic specification and proof techniques with implementations expressed in Pascal and specified in part with inductive assertions. Although DTVS has been developed as a separate system from XIVUS, we plan to combine the two systems in the very near future. This should also provide a good environment in which to experiment with abstract model specifications, which we have already begun to use in some experiments with DTVS.

SUMMARY

We have been using XIVUS and DTVS to verify various programs expressed in a high level language. More importantly, they serve as extremely useful vehicles for experimenting with new verification strategies and techniques, including several methods for structuring and specifying programs. These ideas, well supported by concerns and results from the area of programming methodology, appear to provide the needed separation of concerns in verifying real programs. Our goal remains the construction of a

generally useful set of verification tools that could be widely used in the routine construction of quality software.

REFERENCES

[DoD77] "Department of Defense Requirements for High Order Computer Programming Languages--'Ironman'," January 1977.

[Guttag76] Guttag, J. V., E. Horowitz, and D. R. Musser, *Abstract Data Types and Software Validation*, USC/Information Sciences Institute, ISI/RR-76-48, August 1976.

[Guttag77] Guttag, J. V., J. J. Horning, and R. L. London, *A Proof Rule for Euclid Procedures*, USC/Information Sciences Institute, ISI/RR-77-60, May 1977. Also *Proceedings of IFIP Working Conference on the Formal Description of Programming Concepts*, St. Andrews, Canada, August 1977, E. J. Neuhold (ed.), North-Holland (forthcoming).

[Lampson77] Lampson, B. W., J. J. Horning, R. L. London, J. G. Mitchell, and G. J. Popek, "Report on the Programming Language Euclid," *SIGPLAN Notices*, 12, 2, February 1977.

[London76] London, R. L., M. Shaw, and W. A. Wulf, *Abstraction and Verification in Alphard: A Symbol Table Example*, Carnegie-Mellon University and USC/Information Sciences Institute Technical Reports, December 1976.

[London77a] London, R. L., "Remarks on the Impact of Program Verification on Language Design," *Proceedings of Workshop on the Design and Implementation of Programming Languages*, Lecture Notes in Computer Science, Springer-Verlag, 1977 (forthcoming).

[London77b] London, R. L., J. V. Guttag, J. J. Horning, B. W. Lampson, J. G. Mitchell, and G. J. Popek, *Proof Rules for the Programming Language Euclid*, Technical Report, May 1977.

[Musser77a] Musser, D. R., "Data Type Verification System," USC/Information Sciences Institute Memo, January 1977.

[Musser77b] Musser, D. R., "A Proof Rule for Functions," USC/Information Sciences Institute Memo, April 1977.

[Popek77] Popek, G. J., J. J. Horning, B. W. Lampson, J. G Mitchell, and R. L. London, "Notes on the Design of Euclid," *Proceedings of a Conference on Language Design for Reliable Software*, *SIGPLAN Notices*, 12, 3, March 1977, pp. 11-18.

[Shaw77] Shaw, M., W. A. Wulf, and R. L. London, "Abstraction and Verification in Alphard: Design and Specification of Iteration and Generators," *Comm. ACM*, 20, 1977 (forthcoming).

[Wulf76a] Wulf, W. A., R. L. London, and M. Shaw, "An Introduction to the Construction and Verification of Alphard Programs," *IEEE Transactions on Software Engineering*, SE-2, 4, December 1976, pp. 253-265.

[Wulf76b] Wulf, W. A., M. Shaw, and R. L. London, "Achieving Quality Software: Reflections on the Aims and Objectives of Alphard," *Carnegie-Mellon Computer Science Research Review*, October 1976, pp. 7-15.

2. INFORMATION AUTOMATION

Research Staff:
Donald R. Oestreicher
Robert H. Stotz
John Collins
John Heafner
Wrenwick Lee
Bob Martin
Larry Miller
Paul Raveling
Leroy Richardson
Jeff Rothenberg
Ron Tugender
David Wilczynski

Support Staff:
Joan Malone
Bonnie Zogby

Contributing Staff:
Vernon Dieter
George Dietrich
Nelson Lucas
Robert Parker
Leo Yamanaka

INTRODUCTION

Although the IA project actually began in the fall of 1973, its roots reach back to a three-week study, conducted on behalf of ARPA, of the military communications on the island of Oahu [1]. This study was initiated at the request of the Secretary of Defense for Telecommunications as a part of a Navy program called COTCO, whose mission was to consolidate and improve communications on Oahu. Until ARPA's involvement, COTCO advocated conventional data processing solutions. The ISI report, which recommended a complete islandwide interactive writer-to-reader message service electrically coupled to AUTODIN (the military's backbone communication system), was submitted by ARPA to DoD, where it excited considerable interest but was generally regarded as too radical to be included in a production system without a better appreciation of its cost and benefits.

Recognizing this, ISI cooperated with ARPA in developing a separate program for military message handling and in making the case to the military establishment for conducting a test of an on-line interactive message service in an operational military environment. In December of 1975 a memorandum of agreement among ARPA, the Navy and CINCPAC was signed calling for an operational test of an experimental message service, to be run at CINCPAC staff headquarters, Camp Smith, Oahu. Three ARPA contractors-- Information Sciences Institute, Bolt Beranek and Newman, and Massachusetts Institute of Technology--were considered as candidates to provide the message service to be used for the Military Message Experiment (MME) program. A competitive evaluation was made to choose the message service best suited for the experiment. In February 1977, ISI's SIGMA system was selected. The MME test, which will be funded jointly by NAVELEX and ARPA, is designated to begin in January 1978 and to run for two years.

On the basis of the ARPANET message system experience, we are confident that the proposed message service will have a high payoff to the military. Not only can formal message preparation and delivery become faster and more reliable, but the processing facilities provided can also be put to new use. One extremely useful application is the retrieval of messages from files. The computer can quickly scan large volumes of traffic to retrieve those messages which meet complex selection criteria. Another example is that with such a service the status of a message is automatically available at all stages from preparation to delivery. Much more detailed accounting and auditing is easy to maintain, providing a better understanding of the basic communication process.

Perhaps the most important contribution of such a system will be that it makes available a secure, informal (off-the-record) message facility. This form of electronic mail is swift and convenient to use, and--unlike the telephone--does not require the simultaneous attention of sender and receiver.

To be a success, an on-line message service must provide the improvements inherent in automation without overly disrupting the traditional patterns and procedures that are known to work. The manual nature of today's message service is somewhat cumbersome, but it is extremely flexible; each command or organization is able to tailor its procedures to its own needs. One of the unique goals of SIGMA is to provide this tailorability.

To adequately support military message handling the organizational structure of the user community must be reflected in this service. For example, the rules about who can access what message files and who can release what messages must be carefully modelled. By definition, formal military message traffic flows between commanders of organizations, even though the messages nearly always originate and terminate at much lower levels. This necessitates special "coordination" or "staffing" procedures on outgoing messages (which require approval up the chain of command) and complicates the distribution of incoming messages. SIGMA is unique in its approach to these problems

It is also necessary that the on-line service be easy to use. It is certainly easier to type "send for coordination" than to hand-carry a draft message around to each coordinator. However, by automating this transmission we are faced with making the use of terminals competitive in ease of use with paper and pencil. Toward this end the IA project has developed a highly responsive CRT terminal which allows local two-dimensional editing and contains special features to facilitate the use of the service. See Fig. 2.1. Through this terminal SIGMA is able to provide scanning and editing aids that are not available on existing message systems. For instance, to facilitate integration of comments and changes from several coordinators, SIGMA offers the ability to compare two versions of the same message on separate windows of the CRT screen.

The ARPANET experience provides ample evidence that computer scientists can use on-line systems effectively with little or no formal training in their operation. There are

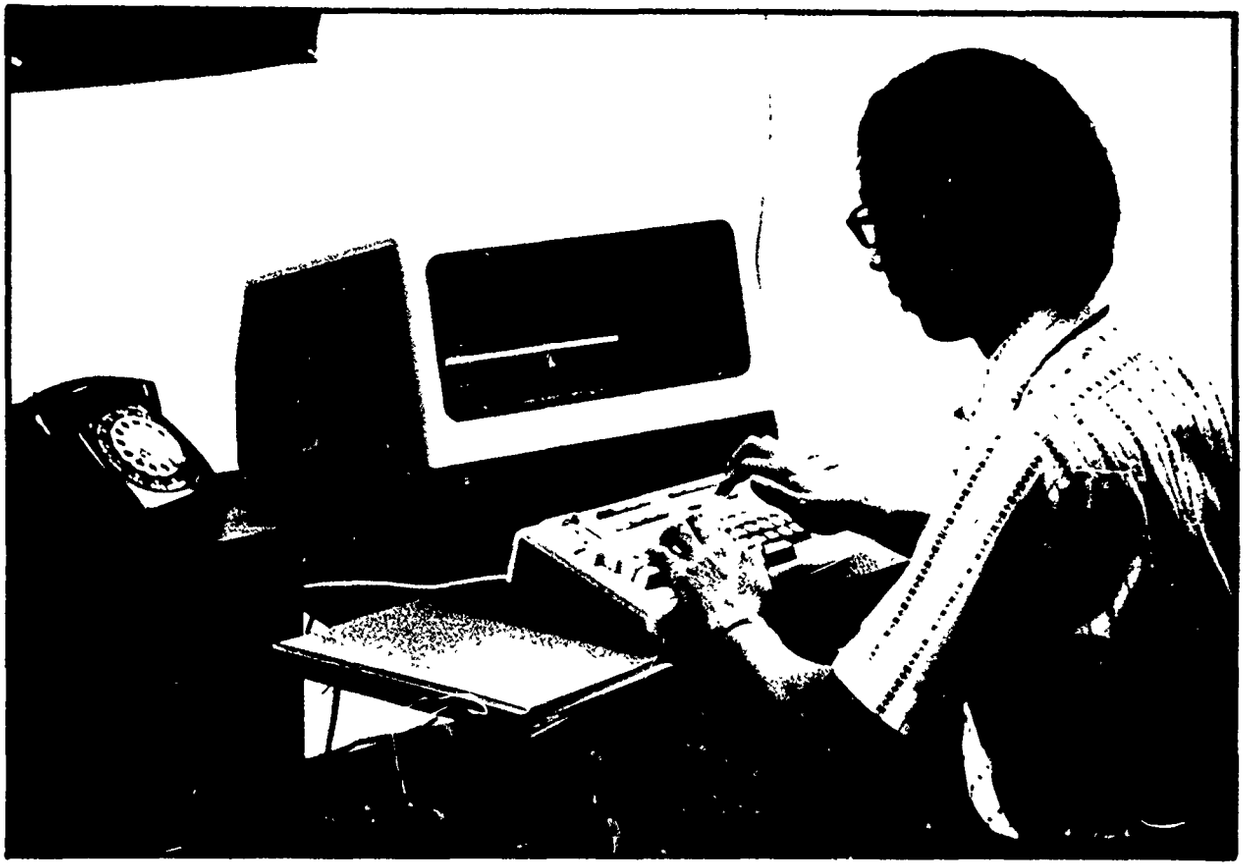


Figure 2.1 The MME terminal environment

also many examples of systems used every day by nonspecialists who have had intensive training (e.g., airline reservation clerks). For a military message service to be effective, however, it must be usable by computer nonspecialists (military action officers) with minimal formal training. Few officers spend more than 10 percent of their time in message-related functions; moreover, the present manual system requires no specialized training. No on-line message service will be used in the military if it is not virtually self-evident and highly supportive whenever the user has any questions or difficulty. The IA project is focusing on this problem as a central research issue.

One component of our research has addressed the user interface, which must be simple to learn and easy to use. In 1974 [4] a methodology was outlined for selecting and improving the user interface language. This methodology (which we call protocol analysis) was tested in 1975 [7] using a collection of computer scientists as subjects, and in February 1976 a protocol analysis test was conducted in Washington, DC using 21 subjects from various naval commands. In July 1976 a similar test was conducted on Oahu using a representative sample (24) of CINCPAC J3 staff members. The results of this test were published [9] and revealed valuable data for design of the user interface.

THE MESSAGE SERVICE

Military message handling may be conveniently divided into two stages: message preparation and message delivery. The preparation stage includes the creation of the draft message and the coordination of this draft with other users until it is signed off for release. For this stage SIGMA provides a special-purpose editing program which understands message formats and checks that the contents of various fields are legitimate. The editor is structured so that a coordinator's editing of a message is stored as a special change file rather than as actual modifications to the original. This facilitates parallel updating of the message by two or more coordinators.

An originator drafts a message, adds comments to fields as desired, fills in the CHOP field with user names, and then sends it for coordination. The message coordinator may then edit the message and add comments of his own. In addition, he may indicate his basic approval or disapproval of the message by "Chopping" Yes or No. Often a coordinator of a message wishes to obtain the opinions of others on his staff before he signs off. SIGMA allows the coordinator to "delegate" to as many people as he wishes the capability to comment on and edit the message (each delegate edits from the original and creates his own change file). The message service retains all of these coordinators' renditions of the message in a single central file, so they may be easily accessed and compared via split screen display.

This coordination process can be iterated as often as necessary, with each version being coordinated independently. A major research goal of IA is to learn more about the staffing process and about how to structure the computer-aided environment to enhance its effectiveness. Eventually an authorized user will RELEASE the message for transmission.

The delivery stage involves conveying the message to its ultimate recipients, archiving it, plus providing aids for the user to sort his messages, scan them, and file them for later retrieval. The first step in this process is to determine distribution for the message. Because of the military policy that all formal traffic flows between commanders of organizations, it is necessary to employ complex procedures to determine the real ultimate recipients.

An incoming message is delivered to a recipient's Pending file (analogous to a person's in-basket or mail box). To see what new messages have arrived, a user asks to DISPLAY his Pending file. He is presented with an index to the contents of the file, an entry for each message. Each entry contains enough information for the user to be able to recognize it, and serves as a reasonably rich context for subsequent retrieval: e.g., the security classification of the message, its Date Time Group, who it is from, its subject, etc. For convenience, a user may attach arbitrary comments to a file entry.

It is military policy to assign one person the responsibility for taking whatever action is appropriate. The officer assigned the "Action" may further delegate the action to a subordinate or may "sell the action" to some other more appropriate officer. The automated message service must keep track of this action assignment until such time as the action has been completed. On the same message "Information" copies are distributed to other action officers who share a need to know. "Readboards" are built, which consist of the accumulation of those messages which should have wide distribution.

SIGMA assists in the distribution function by providing the necessary functions as quick and easy-to-use system commands. Two commands are provided. ACTION is used to assign the responsibility for the message and marks the message accordingly; it causes a citation to the message to be delivered to the action assignee informing him of his responsibility. In addition, a record of the assignment is made in a special "Action Log." FORWARD is used to distribute information copies; it causes a similar citation to be delivered, identifying to the recipient that he has information responsibility on the message. The names of ACTION and INFO recipients are appended to appropriate responsibility fields in the message itself.

A user may put messages into personal or "public" files for later retrieval with the FILE command. Readboards, which are common files that many people may access simultaneously, are built in this manner. This provides the electronic analog of file cabinets. Since the message service can retrieve messages rapidly, these users' "files" actually store only citations to messages, rather than the messages themselves. Eliminating multiple copies limits required computer storage to easily manageable size.

To assist the user in finding messages of interest SIGMA allows the user to select out a subset of messages according to a specified criterion ("selector"). This criterion may be the logical combination of a set of primitive retrieval operations such as who the message is from, character string match in the subject, level of precedence, level of classification, etc. These criteria may be applied sequentially, restricting or augmenting (adding to) the set selected. The user may then store the selection criteria by an arbitrary name for subsequent use. SIGMA also provides tools for specifying "keywords" to received messages, which can be used for later retrieval of the message.

A different form of special handling offered by SIGMA is the alerting mechanism, which will notify the user of the occurrence of certain events by signaling on the user's screen. This occurs immediately, if he is on-line, or as soon as he comes on, if the event occurred while he was off-line. The user's view of SIGMA is illustrated in Figs. 2.2 and 2.3. The screen is divided into three areas. The top three lines tell the user the state of the program, including such information as the name of the objects the user is working with, who is logged on at the terminal, the time of day, etc. The next two lines are the Instruction window where he enters commands to the system. The area below that is the general working space. The message displayed there is a typical AUTODIN message. To the right of the screen are the security lights which indicate the classification of the data on screen.

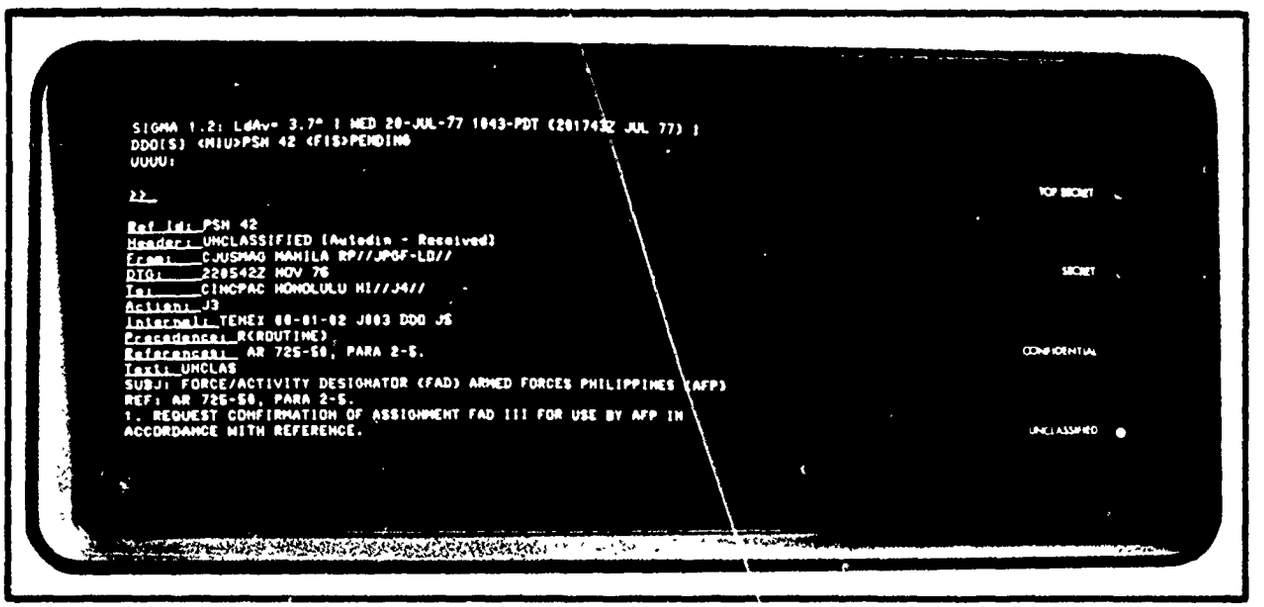


Figure 2.2 MME terminal screen and security panel showing an AUTODIN message

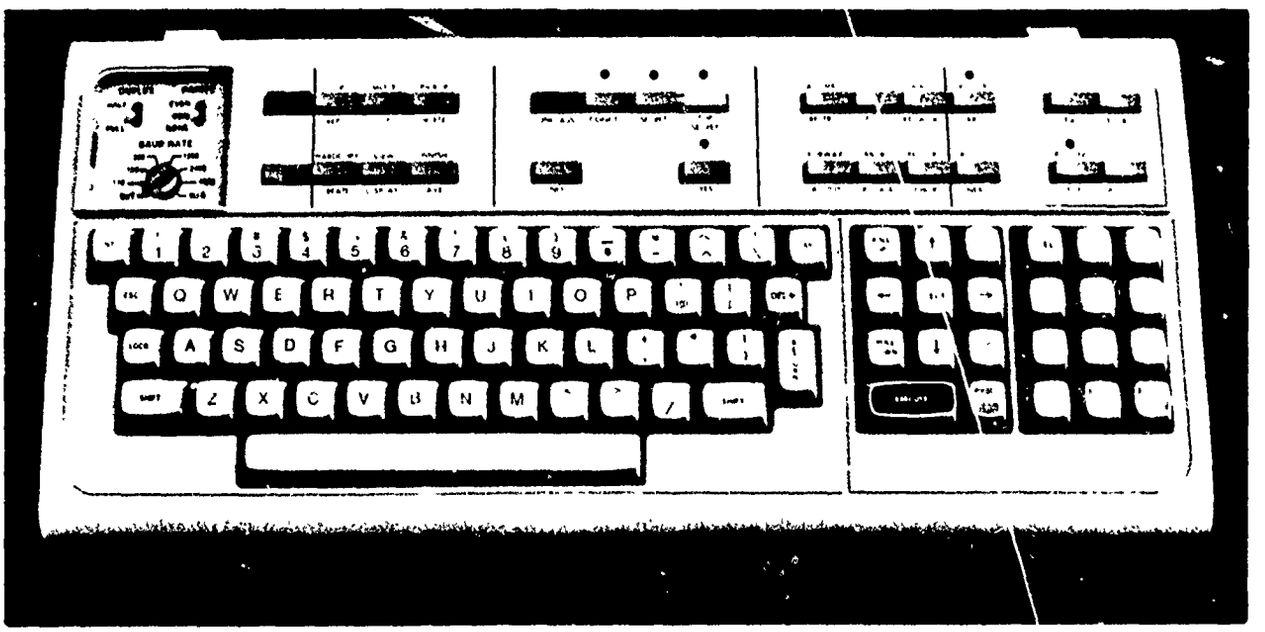


Figure 2.3 MME terminal keyboard

The keyboard has a standard typewriter keyset, a set of terminal control keys to the right, and a set of application oriented function keys above. A removable overlay labels the function keys. The four lights in the center of the function key area indicate the security level of the cursor.

Figure 2.4 shows how the screen looks in different phases of the service. Figure 2.4a illustrates the split screen capability. The Display window, which is the editable area, contains a file. The View window, which is not editable, shows a message from that file. Figure 2.4b shows the HELP system, describing the Create instruction. The user has typed "security" into the New Term field in order to get a description of that facility in SIGMA. In Fig. 2.4c a message in preparation shows the status of the coordination process (J3 has chopped, WREN has not yet read it, etc.). Fig. 2.4d is a file with entries 220 through 228 showing on screen. Entry 226 has a comment attached to it. Entry 224 has been indicated by the user (using the HERE key) to be an argument to the next instruction, "display entry" ("di entry" in instruction window). When the user executes that instruction (by hitting EXECUTE key), the message referred to by entry 244 will be displayed.

SYSTEM DESIGN

User Interface

The approach chosen to provide the necessary support for the user who is not a trained operator or a computer specialist is to interface him to the message service through an "intelligent front-end process" which we call his "Agent." This Agent makes the service appear consistent to the user. It is designed to handle all control procedures (e.g., editing, help, defaulting, error handling, context mechanisms, etc.) with the same code and therefore in the same way throughout all phases of the service. This is a major source of difficulty in the current TENEX message facilities. The Agent and its components are described in detail in Refs. 3, 4, and 5. Briefly, it consists of a Command Language Processor and a Tutor.

Command Language Processor (CLP). This serves as the interpreter for user commands and provides input editing functions and screen control. To support the neophyte, the CLP has a strong emphasis on error detection, recovery, and correction. It also acts as the driver for the rest of the Agent, calling the Tutor when appropriate.

Tutor. This provides intelligent help to on-line users by explaining commands, reporting errors, and providing tutorial, exercise, and reference documentation. On-line lesson material is provided for learning to use the service. At any time the user may enter this tutorial on whatever lesson he wishes. Imbedded within the lessons are exercises, which guide the user through execution of system commands to illustrate their

```
SIGMA 1.2: MyPending:4 : LdAv= 4.7^ ! WED 20-JUL-77 1018-PDT (201718Z JUL 77)
DDO[S] <FIS>PENDING
UUUU:

>>di fi pending

File: PENDING Security: SSS Length: 248
1 CCCC Received  PSN 6003  From: COMUSK SEOUL KOREA//SJ-OF// Subj:
        SCENARIO-MESSAGE-3 (C)
2 CCCC Received  PSN 6003  From: COMUSK SEOUL KOREA//SJ-OF// Subj:
        SCENARIO-MESSAGE-3 (C)
3 UUUU Received  PSN 6001  From: CDR JCRC NAS BARBERS PT HI Subj:
        SCENARIO-MESSAGE-1 (U)
4 SSSS Received  PSN 6004  From: CINCPACFLT MAKALAPA HI Subj:  SCENARIO-MESSAGE
        -4 (C)   Earlier  his year we had some stupefying examples of this
```

(a) split screen

```
SIGMA 1.2: LdAv=12.0^! ! WED 20-JUL-77 1115-PDT (201815Z JUL 77) !
DDO[S] <FIS>DEMO
HELP:  Hit CANCEL key to ge  back to normal screen
                             Current Term: create
                             New Term:

FUNCTION

The Create instruction is used to establish either a file, a message or a text
object, and is also used to save a Selector. These three instructions are
explained below.

The user may define a new file using the instruction form shown below. Files
are defined only in this explicit way.  That is, a file must be defined before
messages are placed into it.  The result of entering a file definition
instruction is to establish a new file containing no entries. The newly
created file is not displayed.  For example, if you are looking at the Pending
File and want to file entries from it into some new file, you can CREATE the
new file without losing your display of the Pending File. You can then FILE
entries from the Pending File into the new file.
```

(b) HELP facility

Figure 2.4 Various [illegible]

```
SIGMA 1.2: LdAv= 4.2v 1 WED 20-JUL-77 1125-PDT (201825Z JUL 77) 1
DDO[S] <MIS>J1 62044Z JUL 77 <FIS>DEMO
UUUU:

>>di entry 8

Ref Id: J3 62044Z JUL 77 (DDO's Version)
Header: SECRET [Autodin in Operation]
Briefing Memo: chop this immediately
From: CINCPAC HONOLULU HI
To: navsecflt
Info: marines
Exempt:
Precedence: R(ROUTINE)
SSIC:
Subject: tank allocation
References:
Text: Provide accountability lists for tank fuel allocations immediately!!!
Downgrade Instructions: GDS
Drafter Chop: DDO - not chopped, TENEX - not read, J301 - not chopped, J3 -
      chop yes, WREN - not read
Drafter Release: WREN - not read
Chop:
Release:
```

(c) message for CHOP

```
SIGMA 1.2: LdAv= 2.5v 1 ...T 20-JUL-77 1039-PDT (201739Z JUL 77) 1
[DO[S] <FIS>PENDING
UUUU:

>>di entry
 20 UUUU Received  220615Z NOV 76 (PSN 38)  From: USDAC TAIPEI TAIWAN Subj:
      PORT VISIT NOTIFICATION -.  CTF 75 300605Z NOV 76 (NOT
 21 UUUU Received  220602Z NOV 76 (PSN 39)  From: FLEWEACEN GUAM Subj: ABEH
      PGTW 220600Z SIGNIFICANT TROPICAL WEATHER ADVISORY 2212 0Z/231200Z NO
      76
 22 UUUU Received  220556Z NOV 76 (PSN 40)  From: COMUSJAPAN YOKOTA AB JA
      Subj: JAPANESE PRESS TRANSLATIONS FOR MONDAY, 22 NOV 76
 23 UUUU Received  220554Z NOV 76 (PSN 41)  From: COMUSJAPAN YOKOTA AB JA
      Subj: AIG 8711 COMPOSITION UPDATE
 24 UUUU Received  220542Z NOV 76 (PSN 42)  From: CJUSMAG MANILA
      PP//JPGF-LO// Subj: FORCE/ACTIVITY DESIGNATOR (FAD) ARMED FORCES
      PHILIPPINES (AFP)
 25 UUUU Received  220526Z NOV 76 (PSN 43)  From: CDR 1ST SIG BDE SEOUL KOREA
      //CCPK-L-M// Subj:  PROPOSED MANUAL OPS CODE (U).
 26 CCCC Received  AMES 913 12 JUN 77  From: AMES Subj: VIEW_MESSAGE
      Sent: Please read this message.
 27 CCCC Received  AMES 91 7Z JUN 77  From: AMES Subj: CLASSIFIED_MESSAGE
 28 UUUU Received  821132Z OCT 77 (PSN 6005)  From: DIA WASH DC Subj:
      SCENARIO_MESSAGE-5 T)
```

(d) file with comment and HERE marker on

functional performance. Exercises operate on a prestored, protected data base, so the user cannot damage real data by mistakenly executing the wrong instruction to the system.

The Agent is designed to collect specific data about the user's use of the service, to gain a better understanding of the system's use in an operational military environment. This data will be useful in determining how well the features of the service are performing their intended functions.

Functional System

The functional aspects of the message service are provided by four modules within each user job (the Functional Module (FM), the Message Access Module, the File Access Module, and the Virtual Terminal Module), and by three free running processes that service all the user jobs (Coordination Daemon, Transmission Daemon, and User Daemon).

The functional aspects of message creation and coordination is handled by the Functional Module (FM) and the Message Access Module of the User Job and the Coordination Daemon. The Access Module deals with the message as it is stored on file by the system. The FM converts the commands from the CLP into appropriate calls on the Access Module and displays the results on the terminal, through the Virtual Terminal (VT). The Access Module shields the FM from the detailed internal representation of the structure of a message. The VT shields the FM from details of the terminal characteristics.

The Coordination Daemon controls the central messages-in-preparation file, where all messages being created or in coordination reside. When a drafter originates a message, or a coordinator adds his changes and comments, this process controls writing the update into the actual message file itself. In addition, the daemon sends out citations to users when they are due to be notified of a message ready for review.

When a message has been reviewed appropriately it is released for transmission. The Coordination Daemon prunes off the excess renditions, all comments, and the coordination list and passes the message to the Transmission Daemon. This process is responsible for formatting and sending the resultant message to its destination. For messages to people outside CINCPAC (AUTODIN messages), this will be through the LDMX, the computer which interfaces to AUTODIN. For messages internal to CINCPAC (formal or informal messages) the Transmission Daemon delivers citations directly itself.

Messages received from the LDMX will likewise be processed by the Transmission Daemon and converted to SIGMA internal form. A special process is dedicated to this task of parsing incoming LDMX messages and converting them into SIGMA's format. Recipients will then be sent citations for these messages and the FM of user jobs will access the message in a similar manner as a message in coordination.

The handling of files is done much like message handling--through the File Access Module that isolates the FM from having to know much about the internals of this file representation. SIGMA files are local to individual users and are controlled by the User Daemon. All other user-specific data is similarly handled by this daemon.

Security

An important requirement for the SIGMA service is that it meet military security specifications. Although this test system will be operated at system high, e.g., access restricted to Top Secret personnel, it is a test objective that the message service address the security issue directly. Previous research done by MITRE has identified the attributes that a system must possess to meet this criterion. The challenge is to build the service in such a way that someone can verify (prove) that the program indeed does have these attributes. The current state of program verification will not handle programs of the size and complexity of SIGMA.

The approach being taken is to concentrate all of the security-relevant code in a single small module (a security kernel). If it can be shown that this kernel does indeed handle all the security issues, the rest of the code does not need to be verified. The MME program schedules do not allow time to actually build and verify this kernel, but its functional performance has been identified, and the system does provide a user interface that responds as though multilevel security were provided; thus the user cannot copy text from a classified message into an object of lower classification without going through a special downgrading operation. The goal of this user interface design is to be true to the security requirements while not repugnant to the user. In the meantime, parallel research by others [13,14,15] is addressing construction of verified security kernels that can be used in future systems.

Privacy (discretionary control of message access on criteria other than security level) is another major concern in a message service. The principal difficulty here is in determining a reasonable statement of what the rules should be. "Need to know" is a highly judgmental quality and very difficult to model. The IA project plans to embody access control mechanisms general enough to be applied to a broad set of models. The service will support author-assigned access control to files, text objects, and annotations associated with messages or files. By applying these controls during the test, we hope to learn what will resolve this complex issue.

Scalability

In the COTCO study it was learned that on the average day on Oahu, 6,000 formal AUTODIN messages are sent out and 15,000 messages are received. To insure that received messages get to the appropriate people, CINCPAC distributes an average of 40

copies of each message. The CINCPAC communication center devotes a 24-hour-a-day printing press to this function, as do other major communication centers on the island. To handle traffic of this magnitude in an on-line system, it is necessary to organize the messages as efficiently as possible. For example, when a message is "delivered," instead of making a private copy for each recipient (as is done with current ARPA message services), SIGMA delivers a brief "citation" to the message. The user is then granted read-only access to the central copy when he wishes to read its contents.

Other SIGMA design decisions also reflect this concern for scalability. The organization of user files is also done by a central process (User Daemon) to compact them as much as possible. In this way, data relevant to many users can be kept in the same TENEX directory rather than requiring a directory per user. The daemons are organized so they may be made into distributed processes that operate across multiple hosts on the network. Thus the service can grow in a straightforward way by expanding the subnet (more nodes and more links) and adding more message processors.

TERMINAL DEVELOPMENT

The cutting edge of the human engineering problem is the interaction at the CRT terminal. We believe it is therefore critical to provide two-dimensional editing with instantaneous feedback of trivial operations, as though the user keystrokes actually performed the operation (insert a character, delete a character, move cursor, scroll, identify a cursor location as being data of interest, etc.). Other more complex operations are dealt with as commands to the system Agent. For these, indication that the command has been input should be instantaneous, but longer delay in actual performance of the action is acceptable.

For a message service test where users are separated from the supporting host by a network, it is difficult to supply the necessary speed of response unless processing is done at the terminal site. With this two-stage architecture in mind, IA has designed a special terminal based on the microprogrammable HP2649 terminal made by Hewlett-Packard. The firmware for this unit has been designed by ISI to make the terminal limitations entirely transparent to the user. The user is never aware that the terminal is continually updating the host computer's model of what is on the screen.

The goal of the MME terminal design is to provide as comfortable an interface as possible for the user. We are trying to provide an environment that competes with what the user has at his desk when he is reading a Readboard. There he has rapid access to many pages and it is easy to switch attention from one document to another. He can capture 4000 characters at a glance, since he sees a full page at once and he can bring up a new page very quickly.

In our CRT we are limited to 24 lines of text (we chose off-the-shelf hardware and felt constrained to keep terminal costs under $10K per copy). To achieve rapid access to more data, we put considerably more memory in the terminal than can be viewed at any one time. The terminal has 12,000 bytes of storage for display data (which is the limit for the HP2649), but can display only 1920 (24 lines of 80 characters). Local scrolling allows the user to rapidly move through the data that is near that which is being displayed. New data is sent to the terminal at the communication rate, which is 2400 baud at this time (we are experimenting with supporting 4800 and 9600 baud as well).

In order to allow multiple objects to be viewed simultaneously, the terminal provides up to 7 "windows". A window may be of arbitrary length, so long as the total of all 7 windows does not exceed the terminal limits (12,000 bytes). The terminal handles all memory management. A window may occupy space on screen ("mapped" onto one or more lines of the screen) or it may be entirely off screen ("unmapped"). The controlling computer assigns this mapping of windows to screen lines. Since the map operation is very fast, the user can switch rapidly between different displayed objects.

An on-screen window normally contains data which is kept in logical margins above and below that which shows on screen. The user can then scroll each window independently. The user selects the window he wishes to scroll by placing his cursor in that window and then pushes the ROLL UP or ROLL DOWN key.

Cursor Movement

Local editing is done through a set of editing keys associated with a cursor. Keys are provided to move the cursor up, down, left and right. In addition there are keys to move right/left by word or by line. These keys move the cursor across window boundaries. To speed up moving between windows UP and DOWN WINDOW keys are provided. These cause the cursor to jump to the last known cursor position in the adjacent window.

Editing

Normal typing keys insert characters at the cursor position and move remaining characters in the line to the right. The terminal automatically wraps overflow onto the next line at word boundaries. Two Delete keys are provided; one deletes the character at the cursor position, the other deletes the character to the left of the cursor (the equivalent of Backspace-Delete). Deleting does not cause "unwrapping" of characters from the next line until the cursor is moved off that line. Holding down SHIFT with the word or line move keys causes word/line delete. Text deletion and insertion never goes outside the window it starts in. If on insertion the cursor reaches the bottom of the window, the contents of the window are scrolled up automatically. No overtype capability is provided, so the user never needs to be conscious of what "mode" he is in.

Domains

Within windows text is organized into sequential blocks called Domains. Domains serve two purposes: they are the logical unit of up to 100 bytes by which all text is passed between the terminal and the host and they provide the application program in the host a means to control the actions of the terminal. Domains have properties, which are assigned when they are created. These properties include certain format control, highlighting (any combination of half or full intensity, blinking or not, inverse video or not), and edit control.

The format controls allow the program to force a domain to start on a new line, or to be indented 1 to 8 tab stops. Edit controls can restrict a domain from being "enterable" (cursor can not enter the domain), "editable" (although the cursor can enter the domain, attempts to add or delete characters will be rejected), or "allow carriage return."

The computer has the facility to have the terminal generate a new domain, change the contents or control of an old domain, split a domain into two, or join two adjacent domains. When the user edits the screen contents, this is reported to the computer as a change to the domain. If new text exceeds the 100 byte limit to the domain, a new domain is created locally by the terminal and the change to the old is reported. This reporting is done completely transparently to the user, who is generally not aware of the domain structure at all.

Communication Protocol

All data transmission to and from the terminal is done in block mode. A special communication protocol insures that these transmissions are successfully received. In essence each transmission is checked for correct protocol sequence, byte count, and data sumcheck, then acknowledged. If there is any difficulty it will be retransmitted.

Security

The standard HP 2649 is slightly modified to provide some external indication of the security of the display contents. Four LEDs are mounted adjacent to the CRT labelled Top Secret, Secret, Confidential, and Unclassified. The application program controls these lights to show the security level of the highest security object on screen. Four LEDs on the keyboard indicate the security of the window in which the cursor resides. The terminal controls this set of lights.

Terminal Production

The Hewlett-Packard 2649 was selected for the MME terminal because of its excellent, flexible control structure and its basic product reliability. Since it was anticipated that the terminal firmware would change during the system development and test, it was essential to use programmable read only memories (PROMs) for the firmware memory. Unfortunately Hewlett-Packard does not offer a control memory board which will accept large (1K byte) PROM memories. As a result, in cooperation with HP, ISI developed its own PROM memory board that was compatible with the HP terminal. Each board holds up to 16K bytes of PROM memory (2708s). Figure 2.5 shows (a) the bare board, (b) the board filled with integrated circuits and sockets for the memory chips, and (c) the completed boards with PROMs inserted. Two boards are required for the MME terminal. Because of high power consumption in the 2708, the PROM board has a special circuit which activates power only on the set of 8 memory chips that is being addressed. Power switches in less than 100 nanoseconds.

Production of MME terminals started with 4 PROM terminals in January 1977, in time for the system evaluation. By the end of June 1977 18 terminals had been delivered. Half of these are at CINCPAC; the rest are at ISI, MIT, BBN and MITRE.

CONTINUING WORK

As described earlier, the focus of the IA project during the previous year has been on the operational test to be run at CINCPAC Headquarters. In addition to the two competitive message services (BBN and MIT), many other people have been involved in the program. BBN has supplied the PDP-10 computer and will operate it for the duration of the test. A team from MITRE and NRL is overseeing the security design of the message service, while another individual from MITRE is working on training. A different MITRE group is developing a Test Plan for the experiment, including determining data collection requirements and analysis to be conducted. The Navy's Test Director is a civilian consultant, while the System Control Officer (SCO) is a Navy Chief Warrant Officer. At this time about six CINCPAC staff members are actively involved in this test on a daily basis. By the time the test begins, one hundred users will be exposed to the message service.

The first seven months of the fiscal year were spent in developing the message service. After the competition in February, the remainder of the fiscal year has been directed to preparing for the operational test. This has included a great deal of planning, installation of SIGMA at CINCPAC, programming to correct the deficiencies and augment the service with additional functions, generating on-line and off-line training material and near continuous on-site (at CINCPAC) assistance.

Since SIGMA was selected for the test at CINCPAC, the overriding goal of the project has been to have the service ready for use by January 1, 1978. This requires an

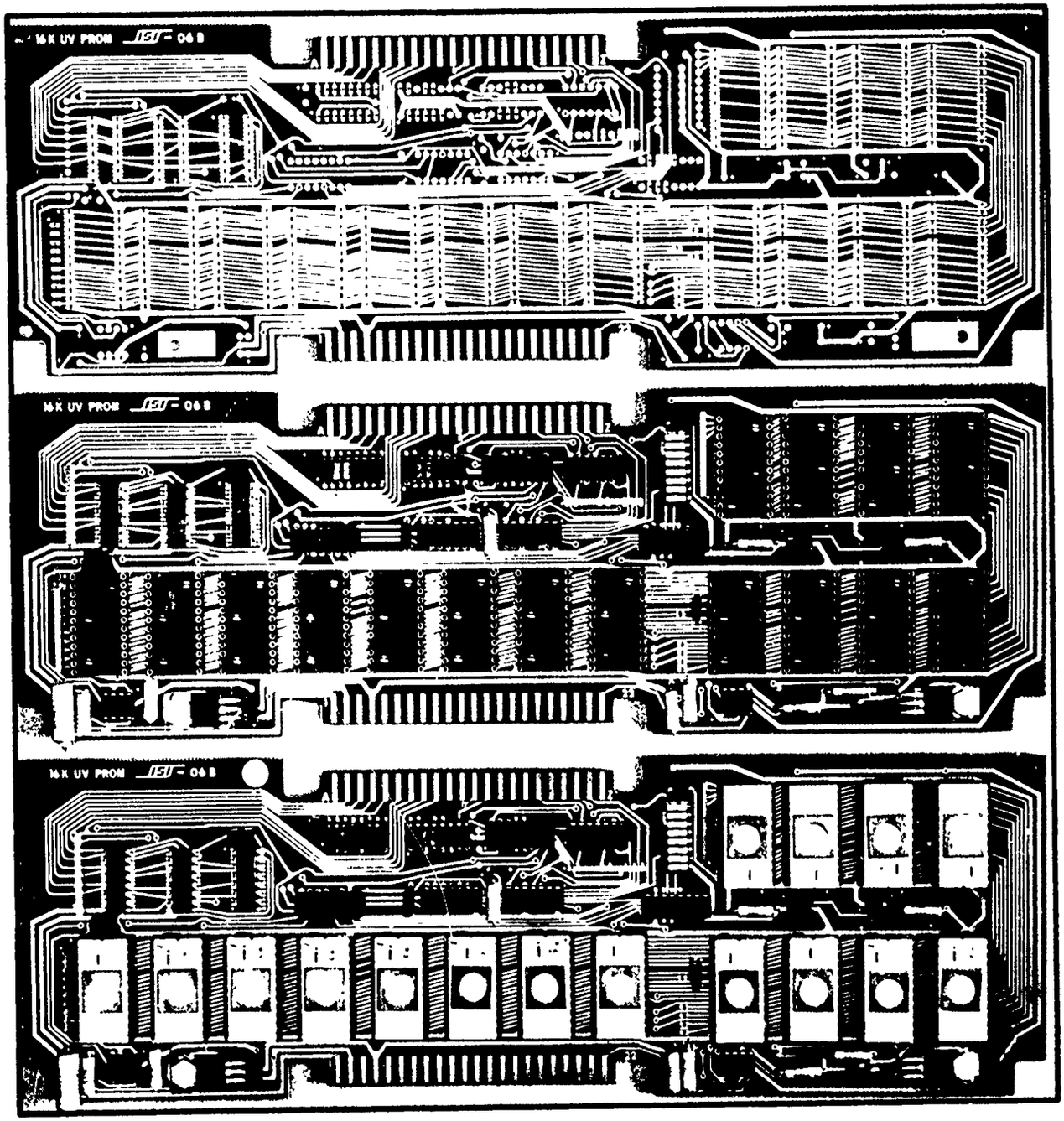


Figure 2.5 The MME PROM printed circuit board, top to bottom: empty board, with components and PROM sockets, and filled with PROMS.

improvement in response time provided by SIGMA and the incorporation of a number of additional functions. Users will be gradually introduced to the system and trained in its use until by December all users will be able to perform their basic message handling on it.

Three milestones are:

1. July 1 - System able to support 5 active terminals and 1 hardcopy unit, and be capable of accepting live traffic from LDMX.

2. Sept. 15 - System able to support 10 active terminals and 3 hardcopy units, and be capable of continuous live traffic from LDMX.

3. Dec. 15 - System able to support 25 active terminals and 7 hardcopy units, accept live traffic from LDMX, and have 3 months data base of messages.

To meet these goals a substantial improvement in system performance must be demonstrated. This improvement is being accomplished by better organizing the functions and streamlining a number of the modules within SIGMA and TENEX. This effort will reduce the size of the program (less paging overhead), reduce the number of TENEX forks required (less interfork delays), and speed up the program execution (less CPU time in execution). In particular the Virtual Terminal (VT) code is being centralized into a single fork with the Terminal Driver, a new faster system routine (JSYS) is being written to be used in place of the String Out operation (SOUT), the Daemons are being rewritten to streamline their operation, and many current foreground functions are being moved into the background, thereby reducing user-perceived delays. The message reception from LDMX is being made faster through more efficient coding. The basic display operations will be made faster by storing preformatted display data for each object. Function key performance is being speeded up by having function keys bypass normal CLP parsing.

Functional improvements being implemented include data collection routines, message retrieval by Date Time Group, message and file format changes, an archive facility, and an access control mechanism applied to files and text objects. Terminal reliability is being enhanced by installation of a new, more resilient communication protocol. In addition diagnostics are being incorporated into the terminal microcode to aid in checkout and maintenance of the units.

When success in the operational test is assured, the IA project will begin to shift its emphasis from developing functional and performance improvements to addressing the many remaining challenging research issues.

REFERENCES

1. Ellis, T. O., Heafner, J. F., L. Gallenson, and J. T. Melvin, *A Plan for Consolidation and Automation of Military Telecommunications on Oahu*, ISI/RR-73-12, May 1973.

2. Tugender, R., and D. R. Oestreicher, *Basic Functional Capabilities for a Military Message Processing Service*, ISI/RR-74-23, May 1975.

3. Rothenberg, J. G., *An Intelligent Tutor: On-line Documentation and Help for a Military Message Service*, ISI/RR-74-26, May 1975.

4. Heafner, J. F., *A Methodology for Selecting and Refining Man-Computer Languages to Improve Users' Performance*, ISI/RR-74-21, September 1974.

5. Abbott, R. J., *A Command Language Processor for Flexible Interface Design*, ISI/RR-74-24, September 1974.

6. Rothenberg, J. G., *An Editor to Support Military Message Processing Personnel*, ISI/RR-74-27, June 1975.

7. Heafner, J. F., *Protocol Analysis of Man-Computer Languages: Design and Preliminary Findings*, ISI/RR-75-34, July 1975.

8. Heafner, J. F., M. D. Yonke, and J. G. Rothenberg, *Design Considerations for a Computerized Message Service Based on Washington, D.C., Navy Personnel*, ISI/WP-1, May 1976.

9. Heafner, J. F., and L. H. Miller, *Design Considerations for a Computerized Message Service Based on Triservice Operations Personnel at CINCPAC Headquarters, Camp Smith, Oahu*, ISI/WP-3, September 1976.

10. Heafner, J. F., L. H. Miller, and B. A. Zogby, *SIGMA Message Service Reference Manual*, ISI/WP-5, February 1977.

11. Oestreicher, D. R., *SIGMA Message Service Security Design*, ISI/WP-8.

12. Heafner, J. F., L. H. Miller, and B. A. Zogby, *SIGMA Message Service Reference Manual*, ISI/WP-5, Version 1.2, June 1977.

13. Neumann, P. G., R. S. Boyer, R. J. Feiertag, K. N. Levitt, and L. Robinson, *A Provably Secure Operating System*, Stanford Research Institute, Project 4332 Final Report 77.2.11.

14. Popek, G. J., and C. S. Kline, *A Verifiable Protection System*, UCLA, ICRS75, 294-304.

15. Bell, D. E., and L. J. LaPadula, *Secure Computer Systems: Mathematical Foundations*, Mitre Corp., MTR-2547, Volume I and II.

3. SPECIFICATION ACQUISITION FROM EXPERTS

Research Staff:
Robert Balzer
Neil Goldman
David Wile

Research Assistant:
Chuck Williams

Support Staff:
Sylvia Meza
Joan C. Nosanov

INTRODUCTION

During the last 25 years of software research two critical ideas have emerged which lay the foundation for major improvements in the software development process. The first is that specification must be separated from implementation; the second is that the interface between these two processes should be a formal operational abstract (i.e., very high level) program rather than the more traditional nonoperational requirements specification. Structured programming represents the first attempt to combine these ideas. It is a special case of a more general two-phase process, called abstract programming, in which an informal and imprecise specification is transformed into a formal abstract operational program, which is then transformed into a concrete (i.e., detailed low-level) program by a series of optimizations (such as choice of algorithms and data representations). Abstract programming thus consists of a specification phase and an implementation (optimization) phase which share a formal abstract operational program as their common interface.

The concept of abstract programming is completed by adding the feedback loops required by testing, maintenance, and tuning. In conventional programming, where no abstract program exists, all of these feedback loops operate on the optimized concrete program. On the other hand, in abstract programming, if an effective method can be found for guaranteeing the validity of an implementation (that is, the functional equivalence of the abstract and concrete programs), then the validation process can be shifted to the specification phase to show equivalence between the user requirements and the abstract program. Thus, validation could, and should, occur before any implementation. Furthermore, if the implementation process could be made inexpensive through computer aids, then maintenance could be performed by modifying the specification and reimplementing it rather than directly modifying the optimized concrete program, as is current practice. The importance of such an advance can be recognized when one realizes that optimization is the process of maximally spreading information (to remove redundant processing), and that modification requires information localization. Thus, the two processes are diametrically opposed; this fact explains much of the current problem with modifying and maintaining existing programs. The second major cause of this dilemma is that optimization obscures clarity and thus makes it difficult for maintainers even to understand how the concrete program operates.

It is therefore clear that major advances in programming will hinge on the ability to provide an inexpensive optimization process with guaranteed validity so that maintenance and validation can occur in the specification phase on the abstract program (as shown in Fig. 3.1) rather than in the implementation phase on the concrete program.

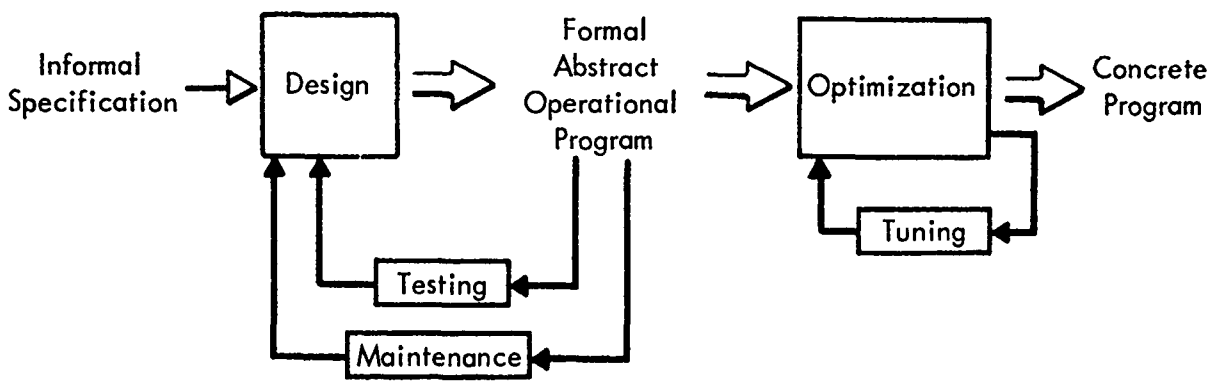


Figure 3.1 Abstract programming

The key element of the whole abstract programming approach is the abstract operational program itself. Currently, considerable effort is directed toward designing appropriate languages for writing such abstract programs; however, no matter how "high-level" these languages become, they are formal programming languages, and as such demand unambiguous, complete, and consistent specifications. It is just these demands that make programming difficult and that make necessary a tool to aid specifiers in building suitable formal specifications.

Currently, all large software systems are specified in written form at a number of levels of detail before implementation begins. One such level is the "functional specification," which describes in natural language how the system is supposed to operate and represents a first-level design. These specifications are currently used by programmers as the description of what to implement. Unfortunately, problems arise because the specifications, although generally understandable, are normally neither complete, unambiguous, nor consistent. The programmer is left to his own ingenuity to discover these problems and to fix them himself or to obtain a clarification from the specifier.

PROJECT GOALS

The Specification Acquisition project has therefore adopted as its goal the development of a system that aids system specifiers (the people who write specifications rather than the people who implement a specification through programming) in converting their informal specifications into a precise operational abstract program. To produce such a system we needed both an understanding of the structure and content of such informal specifications and a theory of how they could be formalized. We therefore first undertook an extensive survey of existing military functional specifications (as embodied in Military Standard-490 B5 specifications). These natural language descriptions represent a first-level design of the intended system. They apportion the required processing into modules and describe the interfaces between the modules and the overall control structure. Because the audience for these descriptions is other people (as opposed to computers), they employ the full variety of detail suppression mechanisms found in natural language, including omitted parameters to actions, anaphoric reference, implicit or omitted control structure, terminology shifts, part/whole interchangeability, etc. (These mechanisms are more fully described in [1]). The reader of the specification is required to amplify the text and determine for himself which details have been suppressed.

Our study of these ..ifications and their detail suppression mechanisms led to our proposed theory of how people understand such specifications and fill in the suppressed details. Our theory is simply that these specifications are understood not in a general natural language context, but rather in the much more specific context that an operational program is being specified. Understanding these specifications basically requires that the correct interpretation of each statement is chosen from several possible interpretations of the natural language statement. The key to our theory is that these choices are guided by the statement's use in the operational program. Fortunately, programs are highly constrained objects (one reason it is so hard to construct them) and therefore act as effective filters of possible interpretations. In Artificial Intelligence terms, program understanding is a domain of strong semantic support.

The SAFE system [2] is based on this theory. It combines the individual statements into a program schema and then attempts to "run" the schema [3]. Symbolic rather than actual data is used as input so that general program behavior can be analyzed. At each step in this "running" of the program a particular interpretation must be chosen for the current statement before it can be executed. The chosen interpretation is accepted if and only if it does not cause any violations of the rules of well-formedness of programs, which are of three forms. First, the program must pass a set of static (syntactic) well-formedness rules such as "parameters must be used in a routine" and "the type of an actual argument must agree with the corresponding formal parameter type." Second, the program must pass a set of dynamic (semantic) well-formedness rules, such as (1) "if X is performed for the purpose of Y then Y must use the results of X," or (2) "the predicate of an IF statement cannot be determinable from the program itself" (if it were, then its evaluation is independent of the actual input and therefore not really a conditional as expected). Finally, the program behavior cannot violate any constraints of the domain.

Whenever one of these rules is violated, the "run" is backed up to the last choice point and a different choice is attempted (if no possibilities remain, then the previous choice point is tried). This process is continued until either a successful "run" is obtained or all possibilities have been exhausted. Interpretations are thus chosen and evaluated in the context of how they are used in the run-time environment of the program. The process of "running" a program with symbolic inputs, called meta-evaluation, has been extensively developed by researchers interested in proving properties of programs. As far as we know, this project represents the first use of meta-evaluation for program understanding rather than program proof.

PROGRESS AND ACCOMPLISHMENTS

This section presents two examples successfully handled by the prototype system. The examples were extracted from actual natural language specification manuals, and the results illustrate the power of the system's context mechanisms. However, our system is a prototype and, as such, it is far from complete. New examples currently expose new problems which are resolved by adding new capabilities to the system. Therefore, until some measure of closure is obtained, it should not be assumed that the prototype will correctly process new examples of the same "complexity." Our goal is to add each new capability in as general a form as possible. In this way we expect to "grow" the system as more complex and varied examples are tried.

The two examples are from a system for scheduling a satellite communication channel by multiplexing it among several users (subscribers). The first example specifies the component of the system which receives a schedule (SOL) from the controller of the satellite channel and extracts from it the portions of the next transmission cycle which have been reserved for a particular subscriber and those portions available to any user (RATS). This information is placed in a transmission schedule used by another component to actually utilize the channel during the allowed periods. Fig. 3.2 gives the actual parenthesized version of the informal input currently used by the system (to avoid syntactic parsing problems). Fig. 3.3 presents a manually marked version of the input which indicates some of the imprecisions which must be resolved to obtain the system's formalization of the specification as an operational program (which is presented in stylized form in Fig. 3.4). In addition to the process description of Fig. 3.2, we have assumed that the formulas referenced and a structural description of the objects of the domain have been separately specified.

The relevant portions of these specifications are that the SOL is an ordered set of subscriber and RATS entries. Each subscriber entry has subscriber identifier and transmission length fields, while a RATS entry has only the latter. The transmission schedule is a set of entries, each of which is composed of an absolute transmission time and a transmission length. One of these entries is the primary entry of the transmission

schedule. Finally, formulas 1 and 2 both take an SOL entry as input and produce a relative and an absolute transmission time, respectively.

```
((THE SOL)
 (IS SEARCHED)
 FOR
 (AN ENTRY FOR (THE SUBSCRIBER))))

(IF ((ONE)
     (IS FOUND))
    ((THE SUBSCRIBER'S (RELATIVE TRANSMISSION TIME))
     (IS COMPUTED) ACCORDING-TO ("FORMULA-1"))))

((THE SUBSCRIBER'S (CLOCK TRANSMISSION TIME))
 (IS COMPUTED) ACCORDING-TO ("FORMULA-2")))

[WHEN ((THE (TRANSMISSION TIME))
       (HAS BEEN COMPUTED))
      ((IT)
       (IS INSERTED)
       AS (THE (PRIMARY ENTRY))
       IN (A (TRANSMISSION SCHEDULE])

[FOR (EACH RATS ENTRY)
     (PERFORM)
     (: ((THE RATS'S (RELATIVE TRANSMISSION TIME))
         (IS COMPUTED) ACCORDING-TO ("FORMULA-1"))
        ((THE RATS'S (CLOCK TRANSMISSION TIME))
         (IS COMPUTED) ACCORDING-TO ("FORMULA-2"])

((THE RATS (TRANSMISSION TIMES))
 (ARE ENTERED)
 INTO (THE SCHEDULE)))
```

Figure 3.2

To transform the informal specification of Fig. 3.2 into the formal operational specification of Fig. 3.4 the prototype system had to provide interpretations for approximately 26 informal constructs. These fall into several different categories, a few of which are illustrated below. The next section discusses the mechanisms by which these categories of informality are handled by the prototype system.

* THE SOL IS SEARCHED FOR AN ENTRY ~~FOR~~ THE SUBSCRIBER.
* IF ONE IS FOUND, THE SUBSCRIBER'S RELATIVE TRANSMISSION TIME ~~IS COMPUTED ACCORDING TO FORMULA-1~~,
* THE SUBSCRIBER'S CLOCK TRANSMISSION TIME IS COMPUTED ACCORDING TO FORMULA-2.
* WHEN THE TRANSMISSION TIME HAS BEEN COMPUTED, IT IS INSERTED AS THE PRIMARY ENTRY IN A TRANSMISSION SCHEDULE.
* FOR EACH RATS ENTRY, THE RATS'S RELATIVE TRANSMISSION TIME IS COMPUTED ACCORDING TO FORMULA-1 AND THE RATS'S CLOCK TRANSMISSION TIME IS COMPUTED ACCORDING TO FORMULA-2.
* ~~THE~~ RATS TRANSMISSION TIMES ~~ARE~~ ENTERED INTO THE SCHEDULE.

Figure 3.3

1. *Incomplete reference.* In informal specifications objects are normally not given an explicit name or label which is later used to unambiguously reference them. Instead, a descriptive reference is used which, since it is incomplete, must be resolved. Sentence 2 contains the pronoun "one" used as an incomplete reference to some object which might be focused. What object does this reference? The previous sentence indicates (after formalization) that the objects being searched for are SOL entries. Hence the pronoun "one" is a reference to a SOL entry.

2. *Missing operand.* An even more extreme case of incomplete reference occurs when the reference is totally omitted. In such cases the specifier assumes that the context is strong enough for the recipient to determine the reference without any explicit clues. Sentence 2 of the example implies that formula 1 should be invoked but doesn't specify what its input should be. However, from

```
(build-transmission-schedule (sol subscriber)
 (CREATE transmission-schedule)
          (search sol FOR A subscriber-entry SUCH THAT
               sid OF subscriber EQUALS sid OF subscriber-entry)
               (IF (locate A subscriber-entry SUCH THAT
                    sid OF subscriber EQUALS sid OF subscriber-entry
                    IN sol)
               THEN
          (MAKE (RESULT-OF (FORMULA-1 subscriber-entry))
               BE THE relative-transmission-time OF subscriber)
               (MAKE (RESULT-OF (FORMULA-2 subscriber-entry))
               BE THE clock-transmission-time OF subscriber))
               (FOR ALL rats WHICH ARE IN sol
          DO (MAKE (RESULT-OF (formula-1 rats))
               BE THE relative-transmission-time OF rats)
                    (MAKE (RESULT-OF (formula-2 rats))
               BE THE clock-transmission-time OF rats))
               (FOR ALL clock-transmission-time OF rats
          DO (MAKE clock-transmission-time BE THE
               transmission-time OF (CREATE transmission-entry))
               (ADD transmission-entry TO transmission-schedule)))
     (WHENEVER (MAKE time BE THE clock-transmission-time OF subscriber)
 DO (MAKE time BE THE transmission-time OF (CREATE transmission-entry))
          (ADD transmission-entry TO transmission-schedule)
          (MAKE transmission-entry BE THE primary-entry OF transmission -schedule))
```

Figure 3.4

the definition of the formula it is known that it takes SOL entry as input, and since we have just found a particular one it is reasonable to conclude that the SOL entry found should be the input to the formula.

3. *Scope of conditional.* In natural language communication the end of a conditional is almost never explicit. Instead, context must be used to determine whether subsequent statements are part of the conditional. In sentence 3 of the example, the input to formula 2 is the SOL entry found in the previous sentence. Thus, sentence 3 is really part of the conditional statement.

4. *Implicit formation of relations.* In sentence 2, the relative transmission time produced by formula 1 is supposed to be associated with the subscriber. Since that association is not established elsewhere, it is implicitly being established here. Hence this passive construct must be treated as an active one.

5. *Implicit creation of outputs.* In a similar fashion, various sentences establish associations with a transmission schedule (the output of this example) but an instance of one is never explicitly created. Such usage indicated that an implicit creation of the output is required.

6. *Expectation failure.* In addition to process and structural statements, a specification normally contains expectations about the state of the computation at some point which provide context for people to explain why something is being done or some properties of its result. They also provide some redundancy against which an understanding of the specification can be checked. In the example, one of these expectations (that all of the components of the entries of the output have been produced) fails, which indicates either a misunderstanding of the specification or an inconsistency or incompleteness. In this case, both our example and the actual specification from which it was drawn are incomplete; they fail to describe how the length field of the entries of the transmission schedule are calculated from the inputs.

The second example from the satellite communication system specifies the component of the system which determines whether the entire text of a message received over the satellite channel should be printed or only its header. This determination is based on the type of message, for which several different cases are specified, and upon whether the message is "addressed" to the subscriber. This, in turn, is determined by seeing if the message contains an addressee which is in one set (the CGL) but not in another (the NP).

The parenthesized informal specification used as the actual input for the second example is shown in Fig. 3.5 and a stylized version of the formal operational abstract program produced by the prototype system is contained in Fig. 3.6.

In addition to the types of informality illustrated above this example also contains the following:

1. *Discovered parameters.* Normally, references to objects within a procedure body are resolved by equating them to some known objects (a parameter iteration variable, or variable determined by a data base pattern match) or by finding a close associatio. between one of the known objects and the reference. When both these methods fail, the prototype system recognizes that it is unable to resolve the reference in the current context. It therefore assumes that the reference must be resolved in some larger context (i.e., the context existing when the procedure is called). It ensures this resolution by making the unresolved reference a parameter of the routine and requiring that it be supplied in each call to that routine. The "subscriber" and "utility-punch" parameters to the top level program are discovered by this means.

2. *Dynamic reference.* Similarly, the inference rule (sentence 6) contains a reference to "subscriber" which cannot be resolved within the context of the inference rule itself. Instead, it must be resolved in the larger context in which the rule is invoked. The inference rule is fired during the compare operation inside of "screen" (which is called by the top level program) when (and if) a match occurs. Hence, its dynamic context is the active portions of those routines. Within this dynamic context, resolving the "subscriber" reference is quite straightforward. It is simply a reference to the "subscriber" parameter of the top level program.

((The CGL) (is) (a list of (addressees))).

((The NP) (is) (a list of (addressees))).

((((Link messages)) ((first-run messages)) and ((normal mode) (rerun broadcast messages))) (are screened) by ((comparing) (all addressees in (the message)) with (the CGL)) in-order-to ((determine) (whether ((the message) (is) ((addressed) to (the subscriber)).

(During ((screening)) (initially (compare) (an addressee) with (the NP))).

(If ((the addressee) (matches)) ((continue) ((screening)) with (the next (message) addressee))).

((Messages with (an addressee ((matching) (a (CGL) entry) (are) ((addressed) to (the subscriber))).

(Context: (((Link messages)) and ((first-run broadcast messages)).

((messages ((addressed) to (the subscriber))) (are noted) so-that ((the message) (will be printed))).

((messages (not ((addressed) to (the subscriber) (are noted) so-that ((the (message heading)) (will be printed))).

((((First-run RS messages)) and ((first-run OLE messages))) (are noted) so-that ((the message) (will be printed))).

(not ((do screen) (((first-run RS messages)) and ((first-run OLE messages)).

(((OLE messages)) (are output) on (the (utility punch))).

Figure 3.5 Narrative message analysis input

```
INFERENCE RULE:
          (MESSAGE-ADDRESSEE ADDRESSEE IN MESSAGE)
        AND(MATCH ADDRESSEE ADDRESSEE#1)
              WHERE (ELEMENT ADDRESSEE#1 CGL)
        IMPLIES
          (ADDRESSED MESSAGE TO SUBSCRIBER)
```

```
SCREEN   (MESSAGE)
LOOP1:   FOR EACH (MESSAGE-ADDRESSEE ADDRESSEE#1 IN MESSAGE)
           DO
             FOR EACH (ELEMENT ADDRESSEE#2 NP)
                 DO   (COMPARE ADDRESSEE#1 TO ADDRESSEE#2 BY EQ)
             IF  (MATCH ADDRESSEE#1 TO ADDRESSEE#3 BY EQ)
                 WHERE  (ELEMENT ADDRESSEE#3 NP)
                 THEN   (CONTINUE LOOP1 WITH ADDRESSEE#4)
                        WHERE (MESSAGE-ADDRESSEE ADDRESSEE#4 IN MESSAGE)
                          AND (SUCCESSOR ADDRESSEE#4 OF ADDRESSEE#1)
             FOR EACH   (ELEMENT ADDRESSEE#5 CGL)
                 DO   (COMPARE ADDRESSEE#1 TO ADDRESSEE#5 BY EQ)
```

```
TOP-LEVEL-PROGRAM (MESSAGE SUBSCRIBER UTILITY-PUNCH)
 IF (IS MESSAGE A FIRST-RUN-MESSAGE) OR (IS MESSAGE A
                          NORMAL-MODE-RERUN-BROADCAST-MESSAGE)
   THEN IF (NOT (IS MESSAGE A FIRST-RUN-RS-MESSAGE) OR
                 (IS MESSAGE A FIRST-RUN-OLE-MESSAGE))
          THEN DETERMINE WHETHER (ADDRESSED MESSAGE TO SUBSCRIBER)
                          BY (SCREEN MESSAGE)
      IF (ADDRESSED MESSAGE TO SUBSCRIBER)
        THEN (MARK MESSAGE WITH MARK*1) FOR PURPOSE (PRINT MESSAGE)
        ELSE IF (NOT (ADDRESSED MESSAGE TO SUBSCRIBER) THEN
             (MARK MESSAGE WITH MARK*2) FOR PURPOSE (PRINT HEADER)
               WHERE (MESSAGE-HEADER HEADER OF MESSAGE)
      IF (IS MESSAGE A FIRST-RUN-RS-MESSAGE) OR
           (IS MESSAGE A FIRST-RUN-OLE-MESSAGE)
        THEN (MARK MESSAGE WITH MARK*1) FOR PURPOSE (PRINT MESSAGE)
      IF (IS MESSAGE A OLE-MESSAGE)
        THEN (OUTPUT MESSAGE ON UTILITY-PUNCH)
```

Figure 3.6

3. *Combination of separate conditional clauses.* Often, in informal descriptions a succession of IF-THEN statements (with no ELSE clause) specify the mutually exclusive actions to be performed for the various cases. Recognizing such mutual exclusion is, of course, critical to a semantic understanding of the specification and is dependent upon determining that the truthfulness of one of the predicates is sufficient to ensure that none of the others is simultaneously true. The prototype system has such a mechanism and is therefore able to combine the actions to be performed when the message is and is not addressed to the subscriber into a single IF-THEN-ELSE statement.

4. *Special action semantics.* We have attempted to remove all representation issues from the specification through our use of the relational data base. This has been highly successful, but some representation issues still remain. The notion of marking or noting something is such a case. Some value (the mark or notation) is associated with an object so that at some later point the existence of the association and particular value will trigger some action. Naturally, the correspondence between a particular value and an action or goal is a representation issue. To remove this problem, the prototype system maintains a set of goals (or purposes) to be achieved at some later time and a set of marking values associated with these goals. When the informal specification specifies a marking operation without indicating the marking value, then the purpose of the marking operation is used by the prototype system to determine the appropriate marking-value. This construct occurs in the three marking operations in this example.

DESCRIPTION OF THE PROTOTYPE SYSTEM

The prototype system is structurally quite simple. It has three phases (linguistic, planning, and meta-evaluation) which are sequentially invoked to process the informal specification. Each phase uses the results of the previous phases, but no capability currently exists to reinvoke an earlier phase if a difficulty is encountered. Hence, either ambiguity must be resolved within a phase or the possibilities passed forward to the next phase for it to resolve.

We will describe the prototype system by working backward from the goal through the phases (in reverse order) toward the input to expose the system design and provide context for understanding the operation of each phase.

The goal of the system is to create a formal operational specification from the informal input, which means that it must complete each of the partial descriptions in the input to produce the output. In general, each partial description has several different possible completions, and a separate decision must be made for each partial description to select the proper completion for it.

Based on the partial description and the context in which it occurs, an a priori ordered set of possible completions is created for each partial description. But one

decision cannot be made in isolation from the others; decisions must be consistent with one another and the resulting output specification must make sense as a whole. Since the output is a program in the formal specification language, it must meet all the criteria for program well-formedness. As we have already noted, programs are highly constrained objects so there are many well-formedness criteria which must be satisfied.

This provides a classical backtracking situation, since there are many interrelated individual decisions that in combination can be either accepted or rejected by some criteria (the well-formedness rules). In such situations, the decisions are made one at a time in some order. After each decision the object (program) formed by the current set of decisions is tested to see if it meets the criteria (well-formedness rules). If it does, then the next decision is made, and so on, until all the decisions have been made and the result accepted. The resulting object (program) is an acceptable solution (formal specification) for the problem (informal specification). If at any stage the partially formed result is rejected, then the next possibility at the most recent decision point is chosen instead and a new result formed and tested as before. If all possibilities have been tried and rejected for the most recent decision point, then the state of the decision-making process is backed up to that existing at the previous decision point and a new possibility chosen. This process will terminate either by finding an acceptable solution (formal specification) or by determining that none can be found.

The order in which decisions (rather than the order of alternatives within a decision) are made should be chosen to maximize early rejection of infeasible combinations of decisions. This requires that the rejection criteria can be applied to partially determined objects. The preferred decision order is clearly dependent on the nature of the acceptance/rejection criteria.

We now let the nature of the well-formedness criteria determine the structure of the prototype system so that the early rejection possibilities inherent in the criteria can be utilized. The criteria fall into three categories: dynamic state-of-computation criteria, global reference criteria, and static flow criteria. Each of these categories must be handled differently.

The dynamic state of-computation criteria are based only on the current "state" of the program and its data base (e.g., "the constraints of a domain must not be violated" and "it must be possible to execute both branches of a condition"). They require that all decisions that affect the computation to that point (but not beyond) must be made before the criteria can be tested. Thus, if decisions could be made as they are needed by the computation of the program and the program "state" examined at each stage of the computation, then the dynamic state-of-computation criteria could be used to obtain early rejection of infeasible decisions.

This is exactly the strategy adopted in the design of the prototype system. However, since no actual input data is available for the program to be tested, and since the program must be well-formed for a variety of inputs, symbolic inputs rather than actual inputs are used. Instead of actual execution, the program is symbolically executed on the inputs, which provides a much stronger test of well-formedness than would execution on any particular set of inputs.

However, completely representing the state of the computation as a program is symbolically executed is very difficult (e.g., determining the state after execution of a loop or a conditional statement) and more detailed than necessary for the well-formedness rules. Therefore, the prototype system uses a weaker form of interpretation, called meta-evaluation, which only partially determines the program's state as computation proceeds (e.g., loops are executed only once for some "generic" element, and the effects of THEN and ELSE clauses are marked as POSSIBLE, but are not conditioned by the predicate of the IF). This meta-evaluation process is much easier to implement and still provides a wealth of run-time context used by the acceptance/rejection criteria to determine program well-formedness.

The global referencing criteria (such as "parameters must be used in the body of a procedure") test the overall use of names within the program and thus cannot be tested until all decisions have been made. They are tested only after the meta-evaluation is complete.

The final category of criteria, static flow (e.g., "items must be produced before being consumed" and "outputs must be produced somewhere"), is more complex. The meta-evaluation process requires a program on which to operate, which may contain partial descriptions that the meta-evaluation process will attempt to complete by backtracking. This program "outline" is created from the informal input for the meta-evaluation process by the flow analysis, or planning, phase, which examines the individual process descriptions and the elaboration, refinement, and modification in the input, then determines which pieces belong together and how the refinements, elaborations, and modifications interact. It performs a producer/consumer analysis of these operations to determine their relative sequencing and where in the sequence any unused and unsequenced operations should occur. This analysis enables the planning phase to determine the overall operation sequencing for the program outline from the partial sequencing information contained in the input. It uses the data flow well-formedness criteria and the heuristic that each described operation must be invoked somewhere in the resulting program (otherwise, why did the user bother to describe it?) to complete the partial sequence descriptions.

If the criteria are not sufficiently strong to produce a unique program outline, the ambiguity must be resolved either by interacting with the user or by including the alternatives in the program outline for the meta-evaluation phase to resolve as part of its decision making process. In the prototype system, the meta-evaluation phase is prepared to deal with only minor sequencing alternatives such as the scope of conditional statements (If a statement following a conditional assumes a particular value of the predicate, it must be made part of one of the branches of the conditional) and demons (Are all situations which match the firing pattern of a demon intended to invoke it or only those which arise in some particular context, and if so what context?). Major sequencing issues--such as whether one statement is a refinement of another or not--that cannot be resolved by the planning phase must be resolved by the user before the meta-evaluation phase.

Both the planning and meta-evaluation phases use a structural description of the application domain to provide context for their program execution, and inference rules which define relation interdependencies in the process domain. This structural base is the application-specific foundation upon which the planning and meta-evaluation phases rest, and must be provided before they are invoked. It contains all the application-specific contextual knowledge. It augments the system's built-in knowledge of data flow and program well-formedness and enables the system to be specialized to a particular application and to use this expertise in conjunction with its built-in program formation knowledge to formalize the input specification.

The construction of a suitable application-specific structural base is itself an arduous, error-prone task. Furthermore, our study of actual program specifications indicated that most of the structural information was already informally contained in the program specification. We therefore decided to allow partial descriptions in the specification of the structural base and to permit such descriptions to be intermixed with the program specification [4,5].

Since we are concerned only with the semantic issues raised by using partial descriptions in the program specification, the system uses a parenthesized version of the natural language specification as its actual input to avoid any syntactic parsing issues. This parenthesized input does not affect the semantic issues we have discussed.

The first tasks, then, of the system are to separate the process descriptions from the structural descriptions, to convert both to internal form, and to complete any partial structural descriptions. These tasks comprise the system's linguistic phase, which precedes the other two.

If a formal structural base already exists for some application, then, of course, it is loaded first and is augmented by and checked for consistency with any structural statements contained within the program specification.

Thus, in chronological order (rather than the reverse dependence order used above), the system's basic mode of operation consists of reading an input specification, separating it into structural and processing descriptions; completing the structural descriptions and integrating the result into any existing structural base; determining the gross program structure by producer/consumer analysis during the planning phase; and, finally, determining the final program structure through meta-evaluation.

PLANS

Having successfully handled several small carefully constructed examples, we want to determine how robust the system is with other reasonable ways of informally specifying these examples. We have therefore constructed about 25 perturbations of the examples to test the system. Several of these we believe the system should handle as is, while others introduce new forms of informality for which new mechanisms must be added.

We have used this example-driven growth technique in the past; it is quite useful in raising those issues which actually occur in informal specifications. We expect that the prototype system will be quite robust on new examples of this size and complexity after completing all the perturbations.

We have also begun to address the issues of how to handle larger, practical-size, informal specifications. Our immediate problem is limited computer address space. This limitation does not constrain the amount of code in our system, but only the amount of data it manipulates. Since our data is largely stored in our relational data base which is currently core-resident, we are implementing a version which is not contained in the address space. Beyond the address space limitations the major size issues are methods for segmenting (modularizing) the specification, utilizing constraints and/or expectations expressed in one module to help understand and validate another, and discovering ill-formed constructs as early as possible and correcting them by special knowledge which avoids overreliance on pure backtracking.

The prototype system is limited not only by size constraints and its degree of robustness, but also by the set of programming constructs it can handle. Foremost among those missing are asynchronous processes and interrupts, which we will begin to incorporate this year.

Although the prototype system suffers from obvious limitations, and is far from being practical, we believe that the basic feasibility of this approach has been demonstrated and that within the next several years a variety of interactive systems which help users formulate formal operational specifications from informal input will become practical and will profoundly affect the way we specify, implement, and maintain software.

REFERENCES

1. Balzer, Robert, Neil Goldman and David Wile, *On the Use of Programming Knowledge to Understand Informal Process Descriptions*, Workshop On Pattern-Directed Inference Systems, Proceedings, Honolulu, Hawaii, May 1977.

2. Balzer, Robert, Neil Goldman and David Wile, *Informality in Program Specification*, USC/Information Sciences Institute, ISI/RR-77-59.

3. Balzer, Robert, Neil Goldman and David Wile, *Meta-Evaluation as a Tool for Program Understanding*, Fifth International Joint Conference of Artificial Intelligence, August 1977.

4. Goldman, Neil, Robert Balzer and David Wile, *The Use of a Domain Model in Understanding Informal Process Descriptions*, Fifth International Joint Conference on Artificial Intelligence, August 1977.

5. Goldman, Neil, Robert Balzer and David Wile, *The Inference of Domain Structure from Informal Process Descriptions*, Workshop On Pattern-Directed Inference Systems, Proceedings, Honolulu, Hawaii, May 1977.

4. NETWORK SECURE COMMUNICATION

Research Staff:
E. Randolph Cole
Thomas Boynton
Stephen L. Casner
John Kastner
James Koda
Eric Mader
Robert Parker
Paul Raveling

Consultant:
Danny Cohen

Support Staff:
Nancy Dechter
Debe Hays
George Dietrich
Oralio Garza
Clarence Perkins
Leo Yamanaka

INTRODUCTION

As more advanced technology is transferred from the laboratory to the military, the need for real-time secure communications of all types has become increasingly severe. Voice, graphics, facsimile, and data of all types must be transmitted quickly and securely. For the past several years, the ISI Network Secure Communications (NSC) group has developed, implemented, and tested systems and methods for voice communication over packet-switched data networks, using the ARPANET as a testbed.

The ARPA NSC effort has achieved its first goal, which was to demonstrate a reliable real-time high-quality packet speech transmission system. This was done using minicomputers, such as ISI's PDP-11/45, as network hosts, and high-speed signal processors, such as ISI's FPS AP-120B, for the necessary speech processing. Such systems are flexible but relatively expensive. Much of the recent and future effort by ISI and other ARPA NSC contractors has been and will be directed toward cost and size reduction, which will greatly ease and accelerate transfer of this technology to the military.

Other future opportunities to apply the knowledge and experience gained in the NSC effort lie in the area of internetting and in the integration of voice, graphics, text, and other media into a highly advanced command and control system. Both these problems are fairly complicated and challenging ones with high payoff.

Most of the NSC project's efforts in the past year have been directed toward implementation, testing, and evaluating systems using the variable frame rate linear predictive coding (LPC) vocoder. This vocoder, called the Phase II LPC system, can transmit good quality speech with an average data rate of about 2000 bits per second. The Phase II LPC system transmits new voice parameters only when the parameters have changed, thus achieving a lower average data rate.

APPROACH

The key feature of the ISI NSC group's approach to packet speech systems has always been to design systems from the user's point of view. Even a research-oriented system will not be fully used if it is difficult to set up, unforgiving of the operator's mistakes, or unreliable. Many computer-based systems of all kinds have ultimately failed because of a poor user interface, regardless of how well-designed the rest of the system was. Two generations of general-purpose hardware control boxes and several fast, flexible, and powerful status displays (using the Hewlett-Packard 2640A CRT terminal) have been designed and implemented at ISI for user interfacing [AR 76].

Once the user interface has been designed, the next step is to design a supporting protocol which is simple, general, and robust. A poor protocol will cause an otherwise good system to fail just as a poorly-designed user interface does. Based on these principles, the ISI NSC group has developed the Network Voice Protocol (NVP), Network Voice Conferencing Protocol (NVCP), the NVP Files Format, and the Packet Speech Measurement Facility (PSMF) Protocol.

The final stage in the system design process is then to design the supporting software package. It is this step which is often overemphasized, since software systems design is probably less of an art than protocol or human interface design. In the past year, considerable effort has gone into a major revision of ISI's packet speech support software. This new structure, which will be described later, represents a quantum step forward in generality, flexibility, and modularity.

The ISI NSC group's approach is therefore a "top-down" or "outside-in" process. These terms, especially "top-down," have been widely used (and abused) recently, but nevertheless such an approach is by far the best one in designing systems which are to be used interactively by a human operator.

Most of the work done to date by the ISI NSC group has been oriented toward building research or development systems. Large-scale integration (LSI) technology has now advanced to the point where the hardware required for a packet speech terminal, including a vocoder, a control panel, and a packet network interface, can be built for less than $10,000. It is reasonable to project that the cost will drop to less than $1,000 within 5 to 10 years. This will make it possible to transfer the packet speech technology to the military, where the need for securable voice communications is becoming acute. Of course, it is more necessary and desirable to include cost considerations in designing packet speech systems than ever before.

RESEARCH AND DEVELOPMENT GOALS

The primary goal of the NSC project at ISI continues to be to provide the best possible research and development in the area of digital voice transmission over packet-switched networks. Work in the past has mostly involved point-to-point or conferencing systems using only voice and only the ARPANET.

The goal of future work will be to expand this effort, using other media along with voice, and other networks, such as the ARPA packet radio network.

Use of voice along with other media such as graphics, text, facsimile, etc. will provide a powerful integrated system for command and control using packet networks. Such a system will help to meet the need for instant, secure, reliable communications that is greatly increasing as military hardware becomes more and more technologically sophisticated.

The possibility of SATNET nodes in the near future will finally provide sufficient bandwidth for large-scale packet speech experiments, such as many-user conferences and high-rate 16 and 32 Kbit vocoding systems.

Other more specific research and development goals for the near future are:

- Packet speech measurements using the PSMF, along with use of the PSMF as an off-line system for storage and retrieval of speech files.
- Specification of a microprocessor-based network controller for packet speech, allowing low-cost connection between a vocoder and a packet network.
- Development of a protocol and software for multisynthesizer conferencing, allowing more than one speaker at a time.
- Implementation and testing of the text-independent voice authentication system developed by Speech Communications Research Laboratory.

PROGRESS

Variable Frame Rate Linear Predictive Coding (VFR LPC)

Coefficient Analysis. The VFR LPC vocoder uses a Markel-type autocorrelation algorithm [MARKEL 76] to produce a set of M vocal-tract parameters and a gain parameter from each frame of input speech. Currently M (the number of poles in the all-pole model of the voice spectrum) is set equal to 9. The vocal tract parameters transmitted are coded versions of the reflection coefficients, k(1) to k(9).

Processing is done on a frame-by-frame basis. The vocoder does a complete analysis frame and a complete synthesis frame during each frame interval. A frame interval is 64 sample periods of 150 microseconds each, or 9.6 milliseconds. The A/D (analog to digital) and D/A (digital to analog) converters for the vocoder are connected via the PDP-11 host computer, to which the FPS is attached. Therefore the PDP-11 serves as the master processor, since it alone has the time information available to it. At the start of each frame, the PDP-11 restarts the FPS, which performs its analysis and synthesis tasks and then halts until the beginning of the next frame.

The FPS uses the 128 most recent input speech samples for its coefficient analysis. At the beginning of a frame, the FPS moves the 64 new input speech samples into a 128-point analysis buffer with the 64 previous speech samples, and preemphasizes the input speech samples using a preemphasis constant of 0.9. That is,

$$y(i)=x(i)-0.9*x(i-1)$$

where the x(i) are the input speech samples and the y(i) are the preemphasized samples. The preemphasis compensates for the normal rolloff of human speech, and is undone by a postemphasis when the speech is resynthesized. The preemphasized samples are then windowed using a 128-point Hamming window. The next step is to autocorrelate the preemphasized, windowed samples. M+1 autocorrelation lags are calculated by direct multiplication and addition. These M+1 points form the rows of a Toeplitz matrix which is then inverted by Levinson recursion [LEVINSON 47] to produce the reflection coefficients. The final result of the coefficient analysis is a set of M reflection coefficients k(1) to k(M), and the zero-lag autocorrelation coefficient r(0). In this implementation, $\sqrt{r(0)}$, the rms energy of the windowed input speech, is transmitted as the gain parameter rather than the usual gain parameter σ, which most LPC vocoders transmit. This is because of the normalized synthesis filter which this implementation uses. It can be shown that if G is the gain of the normalized synthesis filter, then

$$\sigma = G*\sqrt{r(0)},$$

and the output energy is the same as if the usual gain parameter σ had been transmitted and a unity-gain synthesis filter used.

Pitch Analysis. The major component of this vocoder (and many other vocoders) is its pitch period analysis method. As Table 1 shows, pitch analysis not only consumes about twice as much running time as coefficient analysis, but requires more than twice as much program memory. Moreover, pitch accuracy can be the primary factor which determines the subjective quality of the vocoder. A vocoder with an accurate pitch detector can sound much better than one with a poor pitch detector, even though the latter may have a superior vocal tract representation. The pitch analysis used by this implementation is Markel's SIFT method [MARKEL 72], which first lowpass filters and downsamples the input speech to save computing time, derives the prediction error series,

Table 1
TIME AND SIZE SUMMARY
(FPS Variable Frame Rate LPC Implementation)

	Running Time (microseconds per 9.6ms frame)	Program Size (memory locations)
ANALYSIS		
Coefficient Analysis		
Window & preemphasis	132	12
Autocorrelation (10 lags)	343	16
Levinson matrix inversion	224	99 (includes divide)
Main program	4	26
Pitch Analysis		
Lowpass filter	416	17
Window & preemphasis	128	--
Autocorrelation (5 lags)	159	--
Levinson matrix inversion	59	--
Inverse filter convolution	103	15
Autocorrelation (40 lags)	661	--
Pitch extraction heuristics	50	81
Main program	151	102
Coefficient Encoding	100	49
Variable Frame Rate Algorithm	50	55
Buffer Handling	125	39
Analysis Total	2765	511
SYNTHESIS		
Interpolation	216	91
Excitation (main program)	311	104 (includes SQRT)
Synthesis Filter	768	18
Coefficient Decode	30	48
Interframe Save/Restore	26	77
Synthesis Total	1351	338
Grand Total	4.116 ms	849 locations

(Some of these times vary depending on the input and output speech)

and then examines the autocorrelation of the prediction error to find the pitch period and decide whether the speech is voiced or unvoiced.

For pitch analysis, 256 points (38.4 ms) of input speech are lowpass filtered using the normalized synthesis filter with a fixed set of coefficients which implement an elliptical-type lowpass filter. The filtered speech is then downsampled by a factor of 3

and the resulting 85 or 86 points are preemphasized using a preemphasis constant of 1.0 (i.e., a first difference), windowed using a Hamming window, and then autocorrelated to produce five lags, r(0) to r(4). The corresponding set of linear equations is solved by Levinson recursion to produce 4 prediction coefficients, a(1) to a(4) (a(0) is always 1). The 85 or 86 filtered and downsampled points are inverse filtered using the prediction coefficients to form the prediction error series, which should be largely an impulse train spaced at the fundamental pitch period. The first 40 lags of the autocorrelation of the prediction error series are computed, and a heuristic program scans that autocorrelation to find the peak values. The pitch heuristic program then uses the current peak and the peaks from the two previous frames to determine the final estimate of the pitch from two frames ago. Thus the pitch estimate lags 2 frames behind the coefficient analysis. This requires that two frames of vocal tract parameters (reflection coefficients and gain) be buffered at all times.

Synthesis. As has been mentioned, the synthesis uses Markel and Gray's normalized filter [MARKEL 76]. A normalized filter has exceptionally good roundoff noise properties and is therefore very well suited to an integer LPC implementation, although it requires as much as 4 times as many multiplies as other types of synthesis filters. In addition, the gain of the normalized filter has been utilized as described above. Since this floating-point vocoder communicates over the ARPANET with integer LPC implementations, the normalized filter, though inefficient, is still used.

The synthesis filter operates on a point-by-point basis within each frame, and uses a pitch period counter and the voiced/ unvoiced flag to determine when to input a scaled impulse to the synthesis filter (during voiced speech) or scaled pseudo-random noise to the synthesis filter (during unvoiced speech). It is generally necessary to update the synthesis filter coefficients more often than once per frame, in order to generate smoother-sounding output speech. This is done by linear interpolation. Interpolation can be done either time-synchronously (at specified unchanging points within each frame) or pitch-synchronously (once every time a new pitch pulse is input to the synthesis filter). Most vocoders transmit a new set of parameters every frame, and interpolation need only be done between the previous frame's parameters and the current parameters. Since this variable frame rate implementation transmits new parameters only when needed, interpolation over a period of several frames is often necessary.

The Variable Frame Rate Algorithm. At every frame of analyzed speech, a distance measure called a likelihood ratio [MAGILL 73], [ITAKURA 75] is computed and compared to a likelihood ratio threshold (LRT). The likelihood ratio is a measure of the change in the k-parameters (which represent an instantaneous model of the vocal tract) from one speech frame to the next. If the likelihood ratio exceeds the LRT, the change is considered to be significant enough for the vocoder to transmit a new set of k-parameters. If the likelihood ratio is less than the LRT, a bit is sent which tells the synthesizer to simply use the old parameters to synthesize the next frame of speech.

The value of LRT used is the value originally suggested by Magill[MAGILL 73]. Apparent speech quality is changed very little when the LRT is lowered to 0, resulting in parameter updates every frame and a constant output bit rate of over 5000 bits/sec. However, when the LRT is raised, apparent quality seems to degrade quickly, although the speech remains intelligible even at average rates as low as 1500 bits/sec.

Changes in pitch and gain, the other two analysis parameters, are also monitored and compared against individual threshold values to determine if new pitch and gain values should be transmitted. Presently a new pitch is transmitted if the coded value changes at all, and a new gain is transmitted if the coded value changes by more than one step. Obviously, if the coded values do not change at all, retransmission is unnecessary.

In order to track sudden speech transitions, a double threshold scheme has also been implemented. When the change in the parameters exceeds a second, higher, threshold, not only are the latest parameters transmitted, but the previous frame's parameters are also transmitted if they have not been transmitted. This permits the vocoder to track sudden speech transitions more accurately, at the expense of some additional buffering.

Bit Rate and Coding. All parameter encoding and decoding is done by table lookup. Both the transmitter (analyzer) and receiver (synthesizer) have sets of tables which are built in. Encoding in the transmitter is done by a search through the transmitter tables, and decoding in the receiver is done simply by indexing into the receiver table. Pitch and gain are encoded logarithmically using 6 and 5 bits, respectively. Again because of the properties of the normalized synthesis filter, the reflection coefficients are encoded using inverse sine coding, which simplifies integer implementations and is similar to log encoding. Reflection coefficients k(1), k(2) and k(3) are encoded using 5 bits each, k(4), k(5) and k(6) are encoded using 4 bits each, and k(7), k(8) and k(9) are encoded using 3 bits each. Three "flag" bits are transmitted at the beginning of each frame. If the first "flag" bit is a 1, then the a new encoded pitch is included in that frame, if the second "flag" bit is a 1, a new encoded gain is included in that frame, and if the third "flag" bit is a 1, then a new set of 9 encoded k-parameters is included in that frame. Thus if neither pitch nor gain nor the k-parameters have changed enough to be transmitted, a frame consisting of 3 "flag" bits (all of which are 0) is sent. If pitch, gain, and the k parameters have all changed enough to be transmitted, then a frame consisting of 3 "flag" bits (all of which are 1), followed by 6 bits containing the encoded pitch, 5 bits containing the encoded gain, and 36 bits containing the encoded k-parameters is sent. Thus a frame may consist of as few as 3 and as many as 50 bits. Therefore the bit rate of the vocoder can vary from as few as 312 bits/second (although this is unlikely) to as many as 5208 bits/second

Actual bit rate measurements have shown that the vocoder exhibits a long term average bit rate of about 2000 bits/second during periods of continuous speech. The bit rate depends, of course, on the speaker and the amount of background noise present. In the implementation on the ARPANET, where the variable rate is used to best advantage,

the coded gain is also checked to see if the speaker is speaking, and no parameters at all are transmitted if the speaker is silent. This results in an average bit rate that is much lower than 2000 bits/second, depending on the amount of silence within the speech.

Finally, in this implementation the LRT is given to the FPS each frame by the host PDP-11, which therefore has a "throttle" and can control the output bit rate of the vocoder by changing LRT, which also changes the output speech quality. This raises the future possibility of adapting the vocoder to changing network response by allowing quality to vary.

Implementation Issues. Implementation of the VFR LPC vocoder was fairly straightforward. Two of the most common problems in real-time vocoder implementation have been (1) meeting the constraints of fixed-point arithmetic and (2) overcoming the limited program and data memory sizes of early signal-processing computers. Neither of these was a problem in this implementation, because of the floating-point architecture and the large, expandable program and data memories of the FPS.

Experience has shown that in any autocorrelation-type LPC vocoder implementation a very large percentage of the running time is spent in the autocorrelation and synthesis filter, which represent a very small percentage of the total program size. Table 1 shows that this implementation spends 47% (1.93 ms) of its running time executing 4% (34 locations) of the program instructions! Therefore considerable effort was spent on efficient programming of the autocorrelation and synthesis filter subroutines. To carry the point still further, the instruction set of the FPS allows single-instruction loops. The heart of the autocorrelation is just such a single-instruction loop. Therefore, when running VFR LPC, the FPS spends about 25% of its total running time executing a single instruction!

The FPS is more than twice as fast as required for the VFR LPC implementation. This factor eliminated the need to painstakingly squeeze every nonessential microsecond out of the program. Most fixed-rate LPC vocoders use a frame interval on the order of 20 milliseconds. At least 4 separate full-duplex fixed-rate LPC vocoders could therefore be run in real time on the FPS.

Speech quality of the floating-point vocoder implementation seems to most listeners to be slightly, but not strikingly, better than that of a comparable, carefully implemented, fixed-point vocoder. As mentioned before, other factors, particularly the quality of the pitch analysis, can greatly influence the apparent quality of the vocoder, making comparisons difficult at best.

The NVP File Format and the Packet Speech Measurements Facility

During the last year, ARPA set up the PSMF at the Computer Corporation of America (CCA) in Cambridge, Massachusetts. The purpose of the PSMF is to provide a central facility for storage and retrieval of files containing voice data and related information. Such a facility will also greatly assist in the effort to obtain meaningful measurements of the dynamic behavior of packet speech.

In order to utilize such a system, however, there were two major components that had to be defined: (1) a standard format for the data files and (2) a protocol by which to communicate with the file/measurement system. Both of these components were developed by ISI with help from CCA.

The NVP File Format is just that: a format for files of any kind, not just coded speech data, which are to be transmitted, stored, retrieved, or manipulated using the NVP. This allows one participant in a packet speech conference to be not a person, but a program which can store a transcript of the conference. The transcript can later be accessed for listening or editing. Of course, one essential feature is the use of a password for every file, in order to prevent unauthorized tampering with a file.

Format of an NVP file is quite simple but powerful. An NVP file consists of an arbitrary number of "fields," each of which has a 16-bit "field-name" and is of a different "field type." The least significant 8 bits of the "field-name" are the "field type", and the most significant 8 bits indicate the use of the field, depending on the "field type." At present four "field types" are defined: voice, text, binary, and graphics. All of the "fields" are of arbitrary length, and the length of each is written in the field itself.

The PSMF protocol is merely a set of additional commands added to the NVP to allow manipulation of NVP files, and measurement functions based on the contents of the files and the behavior of the network when the files are being received. The set of PSMF messages is:

"OK, I am a PSMF"

"Open a File to Record"

"Open a File for Playback"

"Open a File to Append"

"Retrieve Fields"

"Field to be Recorded/Retrieved"

"Close File"/"End of File"

"Delete File"

"Text to/from Command Parser"

"Data to/from Measurement Facility"

"Open a File for Measurement"

"Positive Acknowledgment"

"Negative Acknowledgment"

Each of these messages is, of course, accompanied by further information such as passwords or data.

The PSMF facility will be fully operational by the end of FY78, and it is anticipated that it will be a valuable part of the ARPA packet speech effort, providing new capabilities and allowing new measurements.

Continuously Variable Slope Delta Modulation (CVSD) Conferencing System

The ISI CVSD conferencing system, described extensively in last year's annual report [AR 76], has undergone extensive use and testing in the past year. Perhaps the most important result was better understanding of the role of "canned" audio feedback in such an environment. The initial use of audio feedback was quite extensive, with prerecorded messages for almost everything, such as "you now have the floor", "you are about to lose the floor", etc. It was found that, wherever possible, a visual indication, such as a light, was much more efficient and less distracting than an audio message. The use of audio feedback was modified accordingly.

In the future, the CVSD conferencing capability will be integrated into the new software structure, with an improved chairman's status display for all kinds of packet speech conferencing, not just CVSD.

SPEECH Communication System

The program SPEECH is a second-generation local and trans-net communication system. It combines features of the previous NVP/NVCP network programs and the previous local CVSD conferencing program to provide a unified base upon which to build future speech experiments. Because this is the second implementation, several structural improvements have been made, based upon previous experience. This allows more efficient division of the task and easier expansion of the program:

- The module structure allows a new vocoder, display, or protocol to be added independently. For example, an existing display device can easily be used with a new vocoder.

- The process structure groups together tasks which are driven by the same events and separates high-rate tasks from low-rate tasks which might conflict with each other.

- Remote vocoders are treated the same as local vocoders to allow network protocol independence.

- Unifying all applications in one program simplifies use of existing capabilities, such as disk record/playback, for future experiments.

SPEECH consists of an Exchange process which supervises all connections among vocoders, and a Vocoder Control Process (VCP) for each vocoder. In Fig. 1, each box represents a process, and the named subdivisions of each box show the modules which contain code executed by that process.

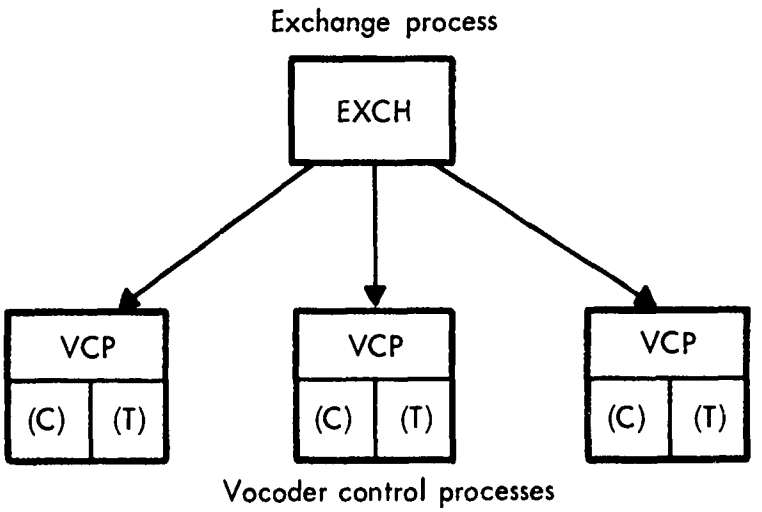


Figure 4.1 Structure of the SPEECH communication system.

The Exchange process establishes and terminates connections, which may be station-to-station or conference calls. This involves vocoder capability matching, creating data structures, etc. For conferences, the Exchange also controls floor switching (speaker selection).

The modularized VCP's provide easy expandability. Each vocoder is identified and controlled by its VCP which executes code in three modules: module VCP, a controller appendage module, and a transducer appendage module. Module VCP contains the data buffer synchronization and distribution code which is common to all vocoders, and it calls subroutines in the controller and transducer appendages to perform functions which are dependent upon the type of transducer (vocoder hardware) and controller (user interface) associated with the vocoder. This structure allows the controller type and the transducer type to be independent. Currently implemented controller types are HP2640A terminal, other standard terminals and general-purpose light-and-switch boxes. The transducer types are CVSD, SPS-41 LPC, and FPS AP-120B LPC. To add a new controller or transducer type, only one module implementing a well-defined set of routines is needed. Since all modules are reentrant, adding another vocoder of a type which is already present requires changing only the configuration definition.

The process structure of SPEECH was chosen to take advantage of the interprocess communication facilities of the EPOS operating system to reduce the size and complexity of the application program. The processes communicate via the SIGNAL and WAIT primitives which transmit from one to six registers of data. For example, to select a particular vocoder as the speaker, the Exchange simply signals the associated VCP, which then performs any actions necessary to start data input from the vocoder hardware and informs the user on his control device that he has the floor. Similarly, when a VCP has a buffer of speech data to distribute to the other vocoders in the connection, it signals the VCP's for those vocoders. The VCP's then play out the buffer and signal back that they are finished. Note that all actions are event-driven so that time is not wasted with polling, and that the information required for an action is immediately available in the context of the process selected to perform that action. Previous network programs grouped together all receiver functions (input from the network and output through the vocoder) and all transmitter functions (input from the vocoder and output to the network). This required both the receiver and the transmitter processes to be activated every ten milliseconds to process a frame of speech to and from the vocoder while both were also processing relatively slower network input/output.

Trans-net communication with a remote vocoder requires no special treatment by the Exchange. A local VCP with controller and transducer appendages to support the desired protocol (e.g., NVP/NVCP) is created to represent the remote vocoder. Data buffers are produced and consumed by a network transducer appendage in a manner analogous to the action of a local transducer. The controller appendage simply translates Exchange signals into control protocol messages and vice versa. This structure allows SPEECH to be protocol-independent. In fact, multiple control protocols can be supported simultaneously, even in the same conference if the data protocols are compatible.

Disk recording/playback of vocoded data is similar to trans-net communication except that control information and data are recorded in, or obtained from, a file instead of the network. Consequently, the disk controller/transducer appendage allows SPEECH to

store data on disk independent of the vocoder type. Vocoder parameters are also stored so that the right type of vocoder can be selected for playback of the file.

The advantage of a modularized and expandable structure was demonstrated by the addition of the capability to access the PSMF at CCA. Although the PSMF is used in a manner different from normal vocoder conferences, with measurement commands and text in the control messages, only minimal modifications were required for the terminal display controller and no changes to the vocoder transducer were required. This represents a considerable savings in effort over a separate implementation of a PSMF program.

Debugger

A breakpoint and trace trap facility has been added to MEND, the EPOS debugger. This facility allows users to temporarily suspend the execution of one or more processes at specific instructions (breakpoints) or after every instruction (trace trap). Two types of breakpoints are implemented: global breakpoints at which all processes stop, and local breakpoints at which only a specific list of processes stop.

MEND breakpoints have proved to be of inestimable value in debugging both application and system software. Because this software makes extensive use of the EPOS multiprocess environment, the ability to have only a specific list of processes stop at a given breakpoint (in a reentrant utility subroutine, for example) has proved to be especially useful.

Hardware Development

The most important hardware development effort was the design, construction, and testing of a A/D and D/A converter system which is interfaced directly to the FPS AP-120B. The new converter system frees the ISI NSC effort from any dependence on the SPS-41. Up to this point, the SPS-41 was still used to emulate a DMA converter system attached to the PDP-11.

Consisting of a 12-bit A/D and a 16-bit D/A converter, the new system uses direct memory access (DMA) capabilities to interface straight into and out of the FPS machine's data memory, without need of assistance from the PDP-11.

Reliability

The primary difficulty experienced with early packet speech systems developed at ISI was poor reliability. The cause of this problem was mostly an unreliable high-speed signal processing computer, which was using over 90% of its capacity while running LPC. The result was a mean time between failures of a few minutes. This unreliability, along with the difficulty of programming, led the ISI NSC group to propose in early 1976 that a more reliable high-speed processor be found and purchased. The proposal was approved, and a Floating Point Systems AP-120B (FPS) was installed at ISI in July 1976.

The Phase I fixed-rate LPC and then the Phase II variable-rate LPC (described elsewhere in this section) were implemented on the FPS in the fall of 1977. In December, a two-week series of daily tests was run to determine the reliability of the FPS-PDP-11/45 packet LPC system. During the entire series of tests, consisting of communication between ISI and CHI (near Santa Barbara) for at least an hour per day at times of peak network load, the ISI system never failed.

In addition, the NSC PDP-11/45 host computer and its software have greatly increased in reliability. These improvements have made tests and demonstrations of the ISI NSC group's work much easier and more meaningful.

IMPACT

The ARPA NSC effort is already beginning to have an impact on military plans for future secure communication systems. One of the tasks given to the DoD's Narrowband Voice Consortium was to follow and report on developments, principally those of the ARPA NSC group, in the area of packet speech systems. This impact can only be expected to broaden as ISI technology continues its explosive growth and the electromagnetic spectrum becomes more and more crowded.

Packet speech is going to greatly influence and enhance military command and control systems, which will use military secure packet networks, such as AUTODIN II, to provide instant communications to military decision-makers. The eventual integration of voice, graphics, text, and other media will create an extremely valuable and flexible command, control, and communication system.

Finally, the ARPA NSC effort is greatly influencing efforts which are under way, by ARPA and others, to develop a very low cost (less than $500) secure voice terminal for future military use. This is the next generation of narrowband terminal--much smaller, cheaper, and lighter than the shoebox-sized, 40-pound, 50-watt, $5,000 system developed by the Narrowband Voice Consortium.

REFERENCES

[AR 76] *A Research Program in Computer Technology: Annual Technical Report*, July 1975-June 1976, USC/Information Sciences Institute, ISI/SR-76-6, 1976.

[MARKEL 76] Markel, J. D., and Gray, A. H., Jr., *Linear Prediction of Speech*, Springer-Verlag Berlin Heidelberg, New York, 1976.

[LEVINSON 47] Levinson, N., "The Wiener RMS (Root Mean Square) Error Criterion in Filter Design and Prediction," *J. Math. Phys.* 25, 1947, 261-278.

[MARKEL 72] Markel, J. D., "The SIFT Algorithm for Fundamental Frequency Estimation," *IEEE Trans.*, AU-20, 1972, 367-377.

[MAGILL 73] Magill, D. T., "Adaptive Speech Compression for Packet Communication Systems," *Telecommun. Conf. Record, IEEE Publ.*, 73 CHO 805-2,29 D 1-5, 1973.

[ITAKURA 75] Itakura, F., "Minimum Prediction Residual Principle Applied to Speech Recognition," *IEEE Trans.*, ASSP-23, 1975, 67-72.

5. COMMAND AND CONTROL GRAPHICS

Research Staff:
Richard Bisbey II
Dennis Hollingworth
Elaine Thomas Sonderegger

Consultant:
Danny Cohen

Support Staff:
Joan Nosanov

THE PROBLEM BEING SOLVED

Effective command and control depends upon accurate and timely exchange of information between command levels as well as effective presentation of relevant data. Communication "up" for reporting and "down" for command is required, as well as communication with data bases, either static or dynamically updated. Advanced information processing methods can and should be exploited to manage and augment information exchange, thereby making it easier for military personnel to interpret and respond to events. The cornerstone to effective information exchange is high quality communications, a critical issue in the design of command and control (C2) systems.

Effective communication is crucial for real-time operation in crisis, as the 55 hours of voice conferencing during the Mayaguez incident testify. Simultaneous graphics communication, typically two-dimensional information like maps and charts, can greatly enhance both the quality and the efficiency of other communication forms and provide depth to the information exchange. Furthermore, graphics are probably the most effective and universal way of communicating information.

In cooperation with the Navy, ARPA/IPTO is preparing a C2 testbed (ACCAT) at Naval Ocean Systems Center (NOSC) to explore and demonstrate the use of advanced computer techniques in military applications. There is a recognized need for high-quality graphics input and output in this environment to accommodate effective data presentation and to support high-quality user interaction with C2 programs and data bases.

In support of the testbed, the ISI C2 graphics project is

1. Developing a graphics system that homogeneously supports graphics terminals of widely varying capability, potentially interacting with one or more different programs running on separate computers connected via a C2 communications network.

2. Developing advanced C2 applications for the testbed that fully exploit the graphics medium.

RELATION TO OTHER WORK

The ACCAT effort is highly interrelated, with numerous contractors supplying various components, including data access systems, information retrieval systems, graphics applications, and graphics display systems. The C2 graphics project interfaces with each of these. Graphics application programs such as NOSC's Warfare Effectiveness Simulator (WES) use the ISI-developed graphic system via graphics language subroutine calls to produce the displays. Situation Display, one of the graphics applications developed by ISI, relies on information retrieval components such as SRI's LADDER system to access the information to be displayed. Finally, ISI provides device drivers for specific display devices. Figure 1 illustrates these interfaces.

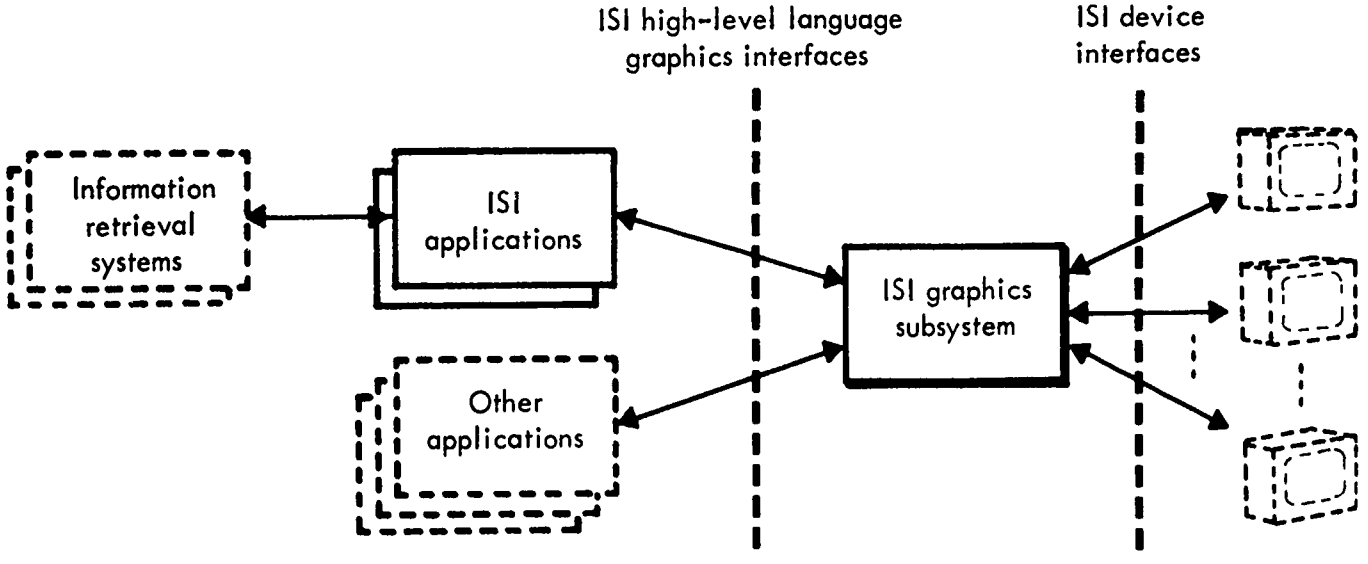


Figure 5.1 Relationship of C2 graphics project to other ACCAT activities

PROGRESS TO DATE

Work on the C2 graphics project was begun in October 1976. Since then, progress has been made in three significant areas: graphics system, map facility, and situation display.

Graphics System

The graphics system requirements of the ACCAT testbed were as follows:

1. The system had to be display-device-independent. In particular, the system had to be able to support both calligraphic and bit-map displays. Advanced graphics features such as color, shading, and input from multiple devices had to be supported in a manner which did not preclude use of the system with devices not supporting those features.

2. The system had to support distributed configurations with application programs physically separated from the display device by a communications network.

The approach adopted was to identify and implement a graphics language comprised of a set of device-independent graphics primitives that could be used to program complicated application-specific graphics interactions. A graphics language (GL) was defined and a first version implemented, providing support for two disparate graphics display devices: a Tektronix 4012 terminal and a Genisco GCT 3000 raster-scan, bit-map color graphics display. The graphics system was made available to the ACCAT facility and was successfully used in several applications, including the situation display and WES.

The graphics system provides constructs for

- Initializing and releasing a display device.
- Defining the device viewport and user coordinate system to be mapped to that viewport.
- Creating, modifying, destroying, displaying, and erasing named segments.
- Generating graphics entities such as lines, dots, text, and arcs (both in relative and absolute form).
- Controlling display characteristics of graphics elements (e.g., line type, intensity, color).
- Accepting data from the terminal.
- Retrieving device/system status information.
- Sending device-specific orders.

- Identifying and filling areas on the display.

The set of graphics primitives provides a core upon which higher level application-tailored graphics packages can be built. For example, a general-purpose parameterized map drawing subroutine was produced that generates maps for the situation display and other application programs.

The graphics primitives in GL are device-independent; the graphics system performs the appropriate feature mapping to support the connected device. For example, an application program may specify the color of a line segment. The system maps the specified color into some suitable color supported by the device, or--in the case of a monochrome display device--into the single color of the device. An inquiry capability is also provided to allow the application program to determine the characteristics of the connected display device and thus exploit the capabilities of the device to their fullest.

The system architecture embodies a functional separation of graphics tasks. For example, the device-dependent elements of the system are logically isolated and can be physically separated from the device-independent elements; support of a new device thus requires only the addition of a new device driver. The device driver software can be separated from the rest of the graphics system by a communications network. This architecture makes it easier to couple software and display devices not developed at ISI to the graphics system.

C2 Map Facility

Some of the application programs of the ACCAT testbed need to display map information. An interim map package was developed to support those applications. The package is callable from Fortran and supports three projection types (Mercator, tangent cylinder perspective projection, and Plate Carrée projection) in two presentation formats. It allows the calling program to specify the center and bounds of the area to be displayed for an arbitrary location on the earth's surface. The Interim Tactical Fleet Command Center (ITFCC) map data base was used as the source for the map data.

Situation Display

ISI developed an interim situation display application both to demonstrate the utility and device-independent characteristics of the graphics system and to explore uses of computer graphics for data presentation for command and control. The application was built by introducing graphics-specific commands into an existing natural language C2 information retrieval system (SRI's LADDER). Figure 2 illustrates the interface between LADDER and the situation display graphics subsystem. Figure 3 shows example output.

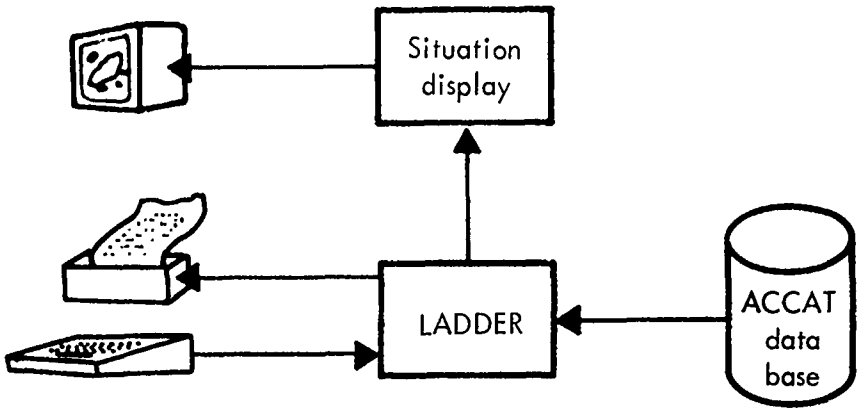


Figure 5.2 Interaction of situation display and LADDER

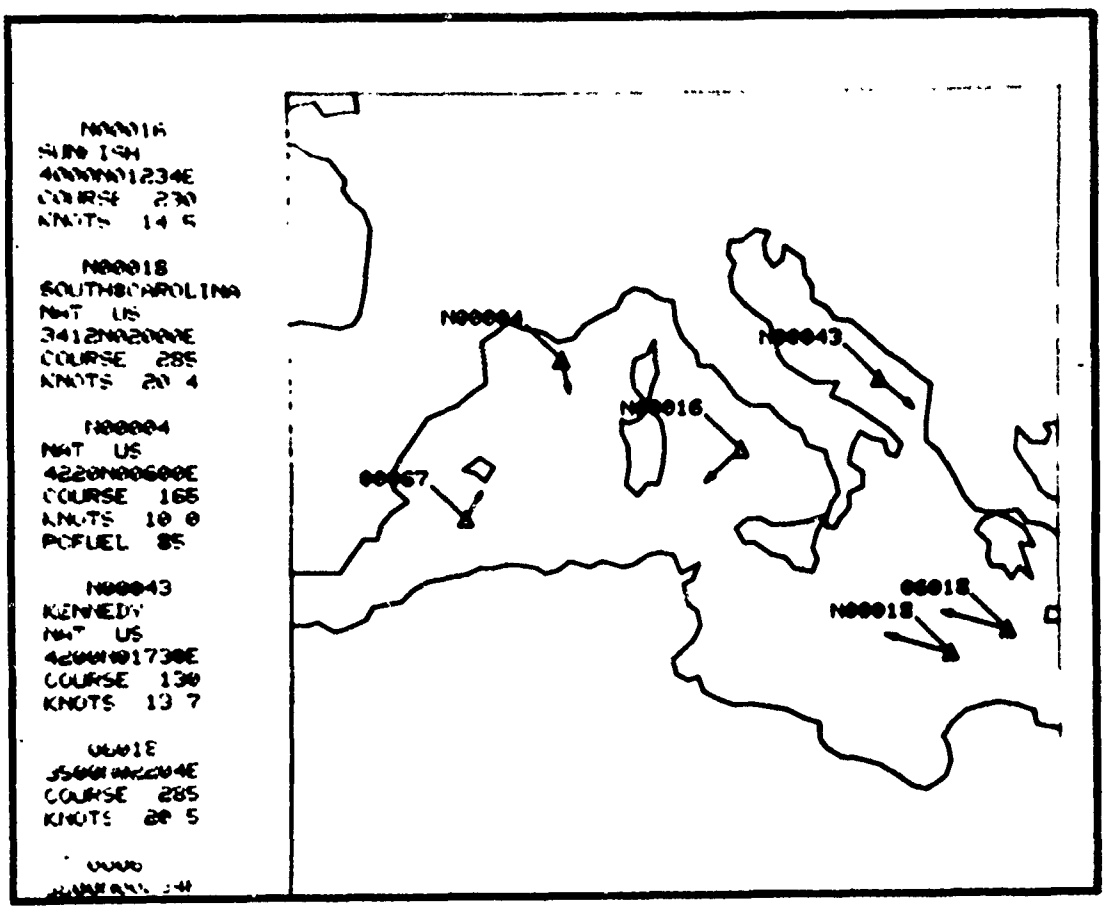


Figure 5.3 Example output of the C2 graphics system

FUTURE RESEARCH AND DEVELOPMENT

Future research and development will focus on the following five areas:

1. Extension and enhancement of the situation display graphics subsystems being developed at ISI.

2. Design and development of a generalized on-line map capability to support command and control.

3. Design and development of a graphic files capability.

4. Design and development of a briefing aid capability.

5. Full implementation, documentation, maintenance, and improvement of GL and GP.

Situation Display

The present situation display application provides a limited means by which information from the ACCAT data base can be graphically displayed. Future activities will involve redesigning and extending the situation display (1) to interrogate and display additional data bases including surveillance data bases and map data bases containing geographic, oceanographic, and meteorological information, (2) to allow the user interactive control of the viewing parameters of the display, and (3) to dynamically update the information displayed.

C2 Map Capability

The present map capability is an interim facility intended to address immediate requirements associated with particular C2 applications. Future efforts will focus on developing an overall ACCAT map strategy that takes into consideration the nature of maps required for C2 processing in general, availability and exploitation of map data from external sources, efficient map data storage schemes, map resolution requirements, and the like. Considerable attention will be given to the issues associated with the management of a distributed map data base, which should accommodate off-line, terminal-accessible map data (e.g., cassette or video-disk storage), standardized maps (e.g., named charts of particular areas), and "custom-made" maps (dynamically generated maps produced from digital data).

Graphics File Capability

For many graphics applications it is important to be able not only to produce displays easily and efficiently, but also to store graphics pictures for retrieval and display at some later time and to communicate graphics pictures between two or more independent programs. The ACCAT testbed environment provides an example of these requirements. An individual preparing a briefing may wish to incorporate graphics pictures generated by a variety of independent programs. The briefer must be able to obtain a machine-readable exact copy of the graphics output of one or more programs, store the copy, transmit it over a digital network such as the ARPANET, and retrieve the stored copy. Graphics files are the underlying mechanism for performing these tasks. ISI will direct part of FY78 resources to designing and implementing a graphics file system that meets the above requirements.

Briefing Aid

An important task in today's C2 environment is for those responsible for the accumulation and aggregation of data to present to their superiors up-to-date information on a specific tactical or strategic situation. Often this involves formally prepared briefings employing films of the situation. Part of ISI's future activities will be oriented toward exploring the use of computerized graphics techniques in improving and augmenting this capability.

ISI will develop the necessary software to support a computer-assisted briefing aid facility. This package will allow an author to compose, review, revise, and present a briefing using computer graphics display hardware. Such a computer graphics briefing will offer the following advantages over more traditional film-based methods: (1) picture preparation and picture presentation may be geographically distributed and (2) the pictures may contain information more up-to-date than that which was available at the time the picture was created.

Graphics Language/Graphics Protocol Development

Although the initial development work for both the GL and GP will be completed in FY77, additional work in several areas will be performed in subsequent years to improve the usability of the graphics system:

Documentation. User-oriented documentation of the GL and GP will be developed and made available. Technical GL and GP documentation which describes and supports the design and implementation decisions will be completed.

Additional CI. functions. A facility for defining and referencing Naval Tactical Display Symbology symbols will be made available.

Remote graphics data capability. A remote graphics data capability which allows picture descriptions to be stored remotely at the display console rather than locally at the application host computer will be developed.

IMPACT

The principal impact of this work is in developing a graphics system architecture that accommodates system decentralization and distributed graphics data storage. Such a system architecture will facilitate graphics/user environments of widely varied display capabilities, storage capabilities, and processing capabilities. An example of such a system is a ship-based graphics system that must interact with and possibly supplement one or more land-based graphic systems associated with large computational environments. The graphics system is intended to provide a sufficiently rich graphics capability to support a wide variety of applications and terminal types.

6. PROGRAMMING RESEARCH INSTRUMENT

Research Staff:
Louis Gallenson
Joel Goldberg
Alvin Cooperband
Ben Britt

Support Staff:
Pamela Kaine
Raymond L. Mason
Rennie Simpson

BACKGROUND

The PRIM (Programming Research Instrument) project was initiated in 1972 to provide an additional level of flexibility in computer architecture for programmers and researchers in the ARPA community by making available a fast microprogrammable computer with appropriate software support to facilitate creating and debugging computer environments (emulators) by remote users via the ARPANET. The system was designed to accommodate many users concurrently employing multiple emulators for generalized hardware/software development (simulations near target-system speeds), direct higher-level-language machine architecture research, and the use and development of specialized logic to enhance existing computers in difficult or time-consuming tasks.

The initial PRIM hardware and software were operational in May 1974 with the ability for remote users to compile and debug microcode. The basic PRIM system, which emphasizes generalized software development tools (with some specific tools), became operational in March 1976.

SYSTEM DESCRIPTION

Detailed descriptions of the PRIM system are available in the previous annual report [1] and in reference documentation [2]. The system is briefly described here, and illustrated in Fig. 6.1. The PRIM system is built in three levels upon a base consisting of the MLP-900 microprogrammable processor and the TENEX timesharing system. First is the *operating-system level* of PRIM, consisting of the hardware and software that support shared access to the MLP-900 from TENEX processes. Second is the *user level*, the PRIM system proper, which provides interactive access to a user at a terminal. And third is the *tool level*, consisting of the set of installed emulation tools, each providing its users with a complete target environment. The user level is for both the emulator developer and the target machine programmer.

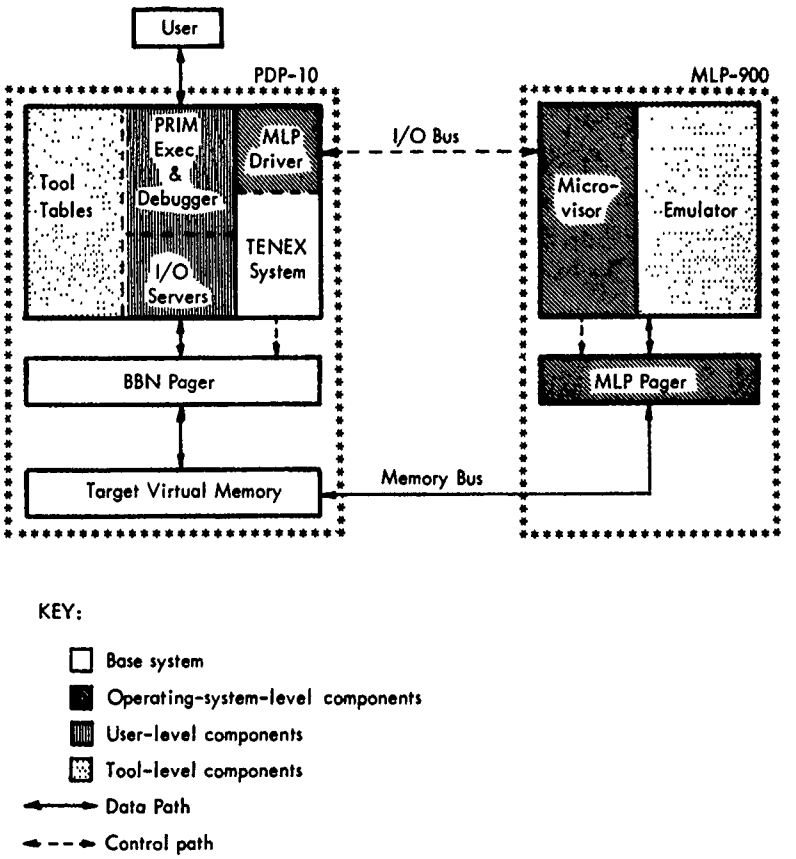


Figure 6.1 Architecture of PRIM

Base System Components

The MLP-900 (Multi Lingual Processor) at ISI is the prototype microprogrammable processor built by Standard Computer Corporation [1]. It is a large, fast, vertical-word processor with a writable control memory. The DEC PDP-10 is a large, general-purpose computer. Connecting new devices to the PDP-10 is a fairly straightforward process, since the I/O bus is extensible and the multiported memory is external to the processor. TENEX is a versatile timesharing operating system developed by Bolt, Beranek and Newman for the PDP-10. It is strongly oriented toward the support of interactive computing and serves both local users and remote users connected via the ARPANET. The TENEX operating system does not respond directly to the user (at a terminal of some kind); rather, it allocates resources, manages the file system, and supports the execution of TENEX processes, each process running in its own paged virtual memory and interacting with its own user in an appropriate manner.

PRIM Operating System Components

The operating-system level of PRIM [3] consists of the hardware connection between the MLP-900 and TENEX, the software necessary to allow TENEX processes access to the MLP-900, and the hardware and software required to guarantee the integrity of TENEX--even against errant microcode. The hardware is a dual state CPU, supervisor and user, within the MLP-900. The software at this level consists of a small operating system resident in the MLP-900, known as the microvisor, and a driver added to the TENEX operating system to control the microvisor and govern access by users (TENEX processes) to the MLP-900.

The MLP-900 is interfaced to PDP-10 main memory and to the PDP-10 I/O bus so that the MLP appears as a PDP-10 I/O device with direct memory access. The memory interface connects the MLP-900 to an existing port (one of four) on the PDP-10 memory. The I/O bus interface allows the exchange of control information between the MLP-900 and TENEX. Via this interface, either processor can interrupt the other.

The net effect of these operating system components is a shared emulation facility in which each emulator runs in its own context (read-only control memory and registers) independent of all others, accessing its own target (virtual) memory. Each such emulation process is completely controlled by the TENEX process that created it. The TENEX process has potential access to all of the context and target memory and may inspect and/or modify them.

User Level PRIM Components

The user level of PRIM consists of an executable TENEX process that defines and implements the PRIM command language. This is the level of PRIM with which the user interacts. Both the emulator developer and the emulator user (target machine programmer) are presented with basically the same command structure. All target-machine-specific data are maintained in tables that are referenced throughout PRIM; there is also a table for the MLP-900 itself for use by the (beginning) emulator developer.

In addition to command interpretation and execution, the PRIM process performs two other functions. First, it contains a module that creates an emulation process for the MLP-900 and controls its execution at the user's behest, thus bridging the gap between the operating system level and the user (at the terminal). Second, it provides an I/O server for that emulation process. There are no peripheral devices attached to the MLP-900; instead emulated devices are mapped onto TENEX disk files and assignable PDP-10 devices (terminals, mag tapes, and network connections).

Microcode is written in GPM, a higher-order machine-oriented language developed at ISI for the MLP-900. There is a GPM compiler available on TENEX as a batch process. The compiler is also used in an interactive mode via PRIM to offer the emulator writer line-at-a-time display and/or modification of control memory.

Tool Level PRIM Components

The tool level of PRIM consists of the set of completed emulation tools that are available to target machine programmers plus an MLP-900 tool available to emulator developers. A completed emulation tool consists of an emulator (pure MLP-900 microcode), a table file describing that tool to PRIM, and a dummy TENEX process by means of which PRIM initializes everything for this target machine. The emulated machine produces better user debugging facilities and greater flexibility in system configuration than the original machine, at the expense of execution speed, while producing bit-for-bit compatible results on all levels of execution. Currently, PRIM also provides the AN/UYK-20 [4], Univac U1050 [5] and the Intel 8080 [6] as completed emulation tools. The MLP-900 tool consists of the GPM compiler and a table file describing the MLP-900.

From the user's point of view, the PRIM system can be in one of three states: the PRIM exec, the PRIM debugger, or the emulated target machine has control. The user's initial interface to the PRIM system is with the PRIM exec. From there he can pass control either to the PRIM debugger or directly to the target machine. From the PRIM debugger, he can pass control either back to the PRIM exec or directly to the target machine. Once control is passed to the target machine it stays there until target execution stops or is stopped by user intervention.

The PRIM exec and debugger have different command languages and provide different services. The PRIM exec has a large number of commands, most of which are used infrequently (Table 1).

These commands all have fixed forms, with only file names and numerical parameter values having a variable content. The command structure adopted for the exec is based on keyword recognition. Because of the large number of keys and the relative infrequency of their use, highly descriptive keywords have been adopted that are easy to remember. The PRIM debugger (Table 2), on the other hand, has a smaller number of commands, most of which are used frequently.

Many of these commands have variable forms and content. The command structure adopted for the debugger is one involving single-key character recognition with complete specification of variables (names or values).

Table 1
PRIM Exec Commands

Configuration	Save/Restore	Miscellaneous
FILESTATUS	LOAD*	CHANGE
INSTALL	RESTORE	CLOSE
MOUNT	SAVE	COMMANDS
PERIPHERALS	SYMBOLS	DEBUG
REASSIGN	TABLES*	ENABLE
REWIND		GO
SET		NEWS
SHOW		NO*
UNMOUNT		QUIT
		TIME
		TRANSCRIPT

* Commands for emulator developers only.

Table 2
PRIM Debugger Commands

Debug Control	Target Control	Display	Storage
COMMENT	BREAK	EQUALS	CLEAR
FORMAT	DEBREAK	EVALUATE	SET
IF	GO	JUMP-HISTORY	MLP-CHANGE*
KILL-SYMBOL	MLP-BREAK*	LOCATE	
MODE	MLP-STEP*	MLP-TYPE*	
NEW-SYMBOL	SINGLE-STEP	NEXT	
OPEN-SYMBOL		PRIOR	
QUIT		SAME	
		TYPE	

* Commands for emulator developers only

Documentation

User documentation has been prepared for each of the existing PRIM tools. In user guides for the specific machine tools (UYK-20, U1050, and 8080), we assume the reader is a familiar user of the specified computer system but requires instructions for interactive code generation and debugging. For the naive programmer a tutorial section provides detailed examples of the utility and power of the PRIM tool. A comprehensive software manual containing machine-independent information has been prepared for the general

PRIM user. The *PRIM System: User Reference Manual* has been prepared for users interested in writing new emulation.

CURRENT ACTIVITY

Goals

The goal of the PRIM project is to provide a state-of-the-art tool for the design, development, and maintenance of software for military computer systems. In particular we are interested in providing interactive software tools more powerful than might otherwise be available for creating and debugging programs. In addition, we have developed and are demonstrating an architecture for providing a generalized emulation facility and service.

Technology Transfer

Since the completion of the basic PRIM system we have concentrated our efforts on transferring this technology into military development projects expressing an interest and need. We have worked closely with the System Design Laboratory (SDL) at Naval Ocean Systems Center (NOSC), with AFDSDC/LGSFE at Gunter AFS, and with the ARPA-sponsored National Software Works (NSW). For each of these development efforts PRIM has been able to provide software tools compatible with the objectives of the user group. This effort entailed tuning the system, adding additional capabilities, and writing documentation to satisfy the stated user needs.

Progress

The System Design Laboratory (SDL) is currently introducing PRIM tools to a user community within the Navy. The charter of SDL is to support the design and development of naval systems employing computers by providing a cohesive set of tools with a common user interface. The AN/UYK-20 [7] is the standard minicomputer for naval systems and the PRIM-UYK-20 is an important part of SDL's initial capability. As selected users were introduced to PRIM, requirements for a basic UYK-20 configuration and available I/O devices became apparent. To satisfy SDL's configuration needs we have extended the UYK-20 capabilities to include MATHPAC (optional microprogrammed Cordic, floating point, and square root algorithms), CIPHER magnetic tape drive, and SI3500 disk. In addition, we have assisted SDL in making UYK-20 support software (MACRO20 assembler, MISG loader, and CMS2 compiler) available as an integrated set of tools under TENEX. Validation of the PRIM UYK-20 has been successfully performed by running the system diagnostics and Level-2 software normally used by conventional UYK-20 systems. The PRIM 8080 emulation was completed at the request of SDL for groups of users at NOSC involved in coding microprocessors. The support software (assembler and compiler) is also available.

The PRIM U1050 emulation package was built in cooperation with personnel from AFDSDC/LGSF at Gunter Air Force Station, Alabama. LGSF is responsible for the maintenance of the air base logistic support system, Standard Base Supply System (SBSS), and PRIM is an important tool in their future development work. The U1050 [8] is an early vintage second generation Univac system produced for business data processing applications. The current PRIM U1050 is configured indentically to the system being used at Gunter AFS. Validation of the PRIM U1050 was successfully performed by running all available system diagnostics and by running the complete SBSS system and comparing the processing of a full day's transactions on both the emulated and real systems.

The PRIM system has been designed to be compatible with the National Software Works (NSW) operating system. NSW provides a convenient user interface and delivery system for the users of PRIM tools. All the PRIM tools and future emulations working under the PRIM exec are compatible with NSW. PRIM has successfully been demonstrated as an NSW tool and provides a unique and powerful service to NSW users.

The PRIM project is also interacting with a research group at the Dahlgren Laboratory of the Naval Surface Weapons Center (NSWC) in developing communication protocols for naval tactical systems. The research requires a flexibly configured UYK-20 on the ARPANET. The PRIM UYK-20 emulator satisfies the current requirements and, as new protocols evolve, can easily be configured to satisfy the future requirements.

RESULTS

The use of PRIM has steadily increased since June 1976, totaling 248 emulation CPU hours during the past 12 months, with a high of 62 hours during October 1976. While some system bugs have been uncovered and corrected during the inital use of PRIM, the MLP-900 continues to operate reliably. The PRIM tools have made significant impact on user development effort. For example,

- PRIM 8080 was used by the IA project at ISI in the successful development of MME/Terminal operating system.

- The PRIM U1050 was successfully validated by personnel of LGS from Gunter AFS, Alabama. During the validation and testing of the U1050, three software bugs in SBSS were detected and corrected; a design limitation in the communications interrupt service routines was also uncovered. These events demonstrated the utility and capability of PRIM to a skeptical user community.

- The PRIM tools (UYK-20, 8080) and software support tools represent most of the initial operating capabilities of SDL. These tools have been well received by the user community and selected users are preparing for their software development within the SDL environment.

- PRIM seminars have been given to USC, Caltech, and UCLA faculty and graduate students. A paper on PRIM has been accepted for the Tenth Annual Workshop on Microprogramming in October 1977 [9].

FUTURE PLANS

Because the MLP-900 is a "one-of-a-kind CPU,"* we have explored alternatives for duplicating the capabilities of PRIM with current-generation commercial systems. The preliminary results of these investigations suggest that existing operating systems and microprogrammable hardware is available to functionally reproduce the PRIM architecture. The system can be designed to completely capture the PRIM exec software. Design and implementation of this system is dependent on appropriate interest and funding from the user community.

PRIM tools will continue to be made available to interested users with access to ISIC and NSW. A minimal effort will be expended for maintenance, user interface, and tuning of the PRIM exec. Additional emulation tools will have to be requested and funded by the interested user. SDL has expressed interest in the implementation of an AN/AYK-14 PRIM tool.

*Standard Computer Corporation is no longer in the business of computer mainframe manufacturing.

REFERENCES

1. *A Research Program in Computer Technology, Annual Technical Report*, Information Sciences Institute, ISI/SR-76-6, June 1976.

2. *PRIM System: User Reference Manual* (in progress).

3. *PRIM System: Tool Builder Manual* (in progress).

4. *PRIM System: UYK-20 User Guide* (in progress).

5. *PRIM System: U1050 User Guide* (in progress).

6. *PRIM System: 8080 User Guide* (in progress).

7. *AN/UYK-20(V) Technical Manual*, NAVELEX 0967-LP-598-1010, Vol. 1.

8. *Univac 1050 Reference Manual*, UP-3912, Univac Data Processing Division.

9. "The PRIM System: An Alternative Architecture for Emulator Development and Use" by J. Goldberg, A. Cooperband and L. Gallenson. Proceedings of the MICRO-10 Workshop, October 5, 1977.

7. PROTECTION ANALYSIS

Research Staff:
Jim Carlstedt
Richard L. Bisbey II
Dennis Hollingworth

Support Staff:
Joan Nosanov

INTRODUCTION

The goal of the Protection Analysis Project is to develop or describe techniques for enhancing the security of existing general-purpose resource-sharing operating systems. These techniques are the means by which errors in the operating system's protection mechanisms can be identified. This problem is of obvious importance in view of the large investment in existing systems, their expected lifetime, and their insecurity. Current general-purpose operating systems, due to their size and complexity, usually contain a large number and variety of errors even after having been in service for years. That these include security errors is proven by the fact that skillful penetration efforts directed against these systems invariably succeed. The task of enhancing the security of such systems is urgent, since many are installed in government, commercial, and military environments with strong security requirements.

No methods exist today for enhancing the security of any large existing software system to the point where it can be proven that no errors exist. Thus the problem is one of attempting to reduce security losses from accidental and intentional violations as much as possible. Security losses will be reduced in proportion to the cost-effectiveness of the available error-finding tools and techniques, as well as their ease of use and applicability to most operating systems.

APPROACH

The Protection Analysis Project and its approach are characterized as follows:

1. Small size. The project has been staffed at an average of about two full-time equivalents, and is currently at the 1.4 FTE level.

2. Near-term results. The project is specifically concerned with results applicable in the near-term future. This means that it cannot engage in research and development efforts that might require many years for completion.

3. Informal methods. Because of the state of research in formal error detection methods, it follows from (2) that the project must concentrate on identifying informal methods.

4. General methods. Because of the variety of models of operating systems in existence, the methods developed or identified must be applicable to a wide class of systems.

5. Static methods. The focus is on static methods, involving the analysis of programs and system documentation, rather than dynamic methods such as testing and auditing of systems in operation.

6. Inexpensive methods. To be cost-effective, security enhancement methods must be applicable without undue expense and their use must lead to the actual discovery of errors. This means that they must be usable by individuals such as system programmers who may be nonexperts in the area of protection in operating systems but who possess a good working knowledge of particular target systems and their supporting machines. A corollary goal is that the error-detection techniques should be automatable to as great an extent as possible.

Within the above dimensions, the essential and distinguishing characteristic of the Protection Analysis Project is its error type or "pattern" approach. Experience has shown that errors are found more effectively during searches for errors of particular well-described types than during nonspecific searches for any type of error. Thus, the large and complex problem of enhancing the security of an operating system can be decomposed into smaller and more manageable subproblems, with possibilities for innovation specialized techniques for each. The pattern approach is discussed in [Carlstedt 75].

The tasks, then, are the following:

1. To identify and describe protection error types.

2. To develop and identify general error detection methods for errors of each type identified.

HISTORY AND RESULTS

The first year of the project was spent in the identification and initial analysis of error types, and in an attempt to formalize their descriptions into "error patterns" to be used as templates to be applied to appropriately "normalized" operating system representations [Carlstedt 75]. This work was primarily empirical rather than theoretical, and was based on a collection of descriptions of protection errors in existing systems of

several major types--TENEX, EXEC-8, MULTICS, GCOS, and OS/360. About a dozen major error types were identified, distinguished primarily by differences in search methods required. (Many errors can be regarded as errors of more than one type, depending on perspective.)

During the past two years work has focused on the analysis and description of a series of error types, together with descriptions of search methods applicable to current operating systems for errors of those types. Three of these were reported last year: *Consistency of Data over Time* [Bisbey 75], *Validation of Critical Conditions* [Carlstedt 76], and *Allocation/Deallocation Residuals* [Hollingworth 76]. An experimental tool was also built to demonstrate the feasibility of automating one of the common normalization tasks, that of determining data dependencies within and across procedure boundaries [Bisbey 76]. The most recent report, near completion, deals with "serialization" errors, those resulting from improper ordering of accesses to shared objects distinct operations [Carlstedt 77]. This report also covers two related error types: those associated with interrupted atomic operations and improper queue management.

IMPACT

The work described here has a significant impact in several areas, most immediately the enhancement of the security of existing operating systems by reducing errors in their protection mechanisms. The empirical and type-structured basis of the research makes it easy to incorporate new error types and error detection techniques should they be identified. These can be used in computer acquisition as part of the evaluation of the security of alternative systems. The base of error and error-type descriptions can also form the basis for a best practices manual for the design of future operating systems and their protection mechanisms. The analysis reported by this project has yielded insights that contribute to a deeper understanding of protection itself. For example, identifying more clearly the concept of criticality (i.e., the most critical objects and operators in a given system) makes it possible to evaluate the particular vulnerabilities of that system more effectively.

REFERENCES

[Bisbey 75] Bisbey II, R., G. Popek, and J. Carlstedt, *Protection Errors in Operating Systems: Inconsistency of a Single Data Value Over Time*, USC Information Sciences Institute, ISI/SR-75-4, December 1975.

[Bisbey 76] Bisbey II, R., J. Carlstedt, D. Chase, and D. Hollingworth, *Data Dependency Analysis*, USC Information Sciences Institute, ISI/RR-76-45, February 1976.

[Carlstedt 75] Carlstedt, J., R. Bisbey II, and G. Popek, *Pattern-Directed Protection Evaluation*, USC Information Sciences Institute, ISI/RR-75-31, June 1975.

[Carlstedt 76] Carlstedt, J., *Protection Errors in Operating Systems: Validation of Critical Conditions*, USC Information Sciences Institute, ISI/SR-76-5, May 1976.

[Carlstedt 77] Carlstedt, J., *Protection Errors in Operating Systems: Serialization* (in progress).

[Hollingworth 76] Hollingworth, D., R. Bisbey II, *Protection Errors in Operating Systems: Allocation/Deallocation Residuals*, USC Information Sciences Institute, ISI/SR-76-7, June 1976.

8. DISTRIBUTED SENSORS NETWORK

Research Staff: Danny Cohen

Consultant: Yechiam Yemini

Support Staff: Debe Hays

INTRODUCTION

This project's objective is to investigate the impact of communication technology on detection technology, and to demonstrate that a distributed network of remote (and possibly different) sensors interconnected by appropriate communications is a cost-effective way to solve the detection problem.

The model used consists of some *space+time* domain in which *targets* are moving. A set of *sensors* is used, each of which is capable of extracting some partial system information. The objective is to extract the *state* of the *system* from the information available from the *sensors*.

Sensors obviously may be of many different types -- active or passive, directional or omnidirectional, of low or high data rate, etc. Some sensors can function independently (such as *radar*) and others may operate only as some array element (such as hydrophones).

In this project we try to capture the features common to any detection problem, and address mainly the sensor-independent issues.

We divide our model into two levels. The "lower" of the two is the level in which row data (or *signals*) are communicated. The other, or "higher", is the level at which features (or *symbols*) are communicated.

We use a model consisting of clusters of sensors, interconnected via a packet radio network (PRNet) with each cluster controlled by its own central controlling unit. These clusters constitute the lower level of the system. Therefore, the lower level of communication is referred to as the "*intracluster-communication*".

In this model the higher level is a network of clusters and processors cooperating with each other, using external knowledge when necessary and feasible. The communication at this level is referred to as the "*intercluster-communication*".

Our project is organized around two distinct vectors, dealing with intracluster and intercluster issues.

INTRACLUSTER ISSUES

We address two intracluster issues:

- The intracluster communication protocol
- Finding the position of each sensor

The main objective of the communication protocol is to support the flow of raw data from each sensor to the central processor in charge of operating this cluster.

In the case of rapidly deployed systems the position of each sensor must be found. Since the PRNet will have a built-in feature for distance measurement (DM) between any units in communication, this information can be used for the solution of the position of each sensor.

This gives rise to our work on solving shapes of bar structures, where only the length of each bar is known. More details of this work are given below.

It is of interest to note that exactly the same technology as the one developed here can be used to locate automatically each unit in some fleet of vehicles communicating by PRNet, such as a fleet of ambulances in peacetime or jeeps at a battlefield.

THE INTRACLUSTER PROTOCOL

The communication protocol we are examining for the intracluster level defines at least the following six classes of messages:

- Data messages
- Synchronization messages
- Connectivity messages
- Distance measurements
- External positioning messages
- Position inquiries

It is too early at this stage to be able to define the grammar and the format of these messages.

Data messages are those used by the central processing unit (the controller) of the cluster to request data and by the sensors to respond to these requests.

The form of these messages depends heavily on the capabilities of the sensors. Examples of messages in this class are:

- HYDROPHONE X, START/STOP SENDING YOUR DATA.
- RADAR Y, ADVISE WHEN YOU HAVE A TARGET WITHIN 20 MILES.
- ARRAY Z, REPORT SPECTRUM FROM f1 TO f2 ALONG AZIMUTH N.

Time synchronization messages are intended for synchronization of clocks. It is not clear yet if this function could be extracted from the distance measurements (*time-stamping*) messages or not, and the exact implementation strategy for this function is not yet clear. However, for many cases it is important to synchronize data from several sensors, and the resolution and accuracy of synchronization is a key parameter for both space- and time-resolution of target detection.

Connectivity messages are those used by the controller to interrogate the full connectivity of the communication network and by the sensors to respond.

An example of a message in this class is: "UNIT X, DO YOU HEAR UNIT Y?"

Distance measuring messages are used to extract distance information along the communication paths. If the hardware of the PRNet elements is modified such that the distance is computed automatically for each communication path, then these messages can be combined with messages of the connectivity inquiries class. If this modification is not implemented these messages will be used for activating the *time-stamping* mechanism along sets of closed paths. By simple arithmetic the total communication delay may be extracted from the time-stamps, in spite of phase differences between the various clocks used by each of the sensors. In order to achieve this phase difference cancellation, only closed paths can be used.

External positioning messages are used by the controller to solicit any additional positioning information ("external") which may be available to some of the units. This is necessary since it is not possible to solve for position from distance only.

An example for a message in this class is "UNIT X, DO YOU KNOW YOUR POSITION?" Obviously, such a message will be sent only if the controller has some reason to believe

that this unit has an access to information about its position, e.g., from a GPS (Global Positioning System).

Position inquiry messages can support users' interfaces and systems where the individual units have a need to know their own position, as computed by the system. Examples of such messages are:

- CONTROLLER, WHERE IS UNIT A?
- UNIT A IS AT POSITION (X,Y).

POSITION SOLVING

The objective is to find the position of all units from a set of known distances between them. The results of this work will contribute to the design and implementation of distance measuring capabilities in the PRUnits, built by Collins.

This position solving problem is equivalent to determining out the shape of a bar structure from the lengths of the bars. Obviously, any solution based only on these distances can be translated, rotated and reflected. Hence, the solution has at least 3.1 degrees of freedom. (Here 3.1 means three continuous and one binary degrees of freedom.)

Therefore, in the original problem, let us assume that the position of one point (i.e., sensor position) is also given, and the direction from it to one of its connected neighbors (i.e., the position of two points with a known distance) is known.

Solving for the position of N points requires 2N pieces of data, in the two-dimensional space. Since there are 3 continuous degrees of freedom, and 3 other known data (a position and a direction) only 2N-3 distances are required.

In the case of a structure with 2N-3 bars, which is "triangulated" (i.e., composed of triangles) there are N-2 triangles, each of which may have either clockwise or counterclockwise sense. Hence, there might be up to N-2 possible discrete symmetries (reflections) to the solution, which leads up to 2^(N-2) possible solutions!

A structure which contains continuous symmetries is defined as *flexible*. There are structures which contain no continuous symmetries, but length measurement errors lead to position errors of a higher order of magnitude (typically square root order). These structures are called *sensitive*. For example, if the position error is the square root of the length error, and this is 1 foot out of 1 nautical mile, then the position error is greater than 77 feet!

Structures with no continuous symmetries which are not *sensitive* are called *rigid*. One can show that in such a structure length measurement errors lead to position errors which are of the same order of magnitude. We will develop algorithms to solve efficiently for positions from the distances and to distinguish the *flexible* and *sensitive* cases from the *rigid* ones.

INTERCLUSTER PROTOCOLS

The object of the intercluster protocol is to support communication at the *symbol* level, in a sensor-independent fashion. This protocol should support both the communication needs of sensor clusters of similar type which communicate in order to increase their geographic coverage, and the communication needs of clusters of different types, communicating in order to increase their detection capabilities by cross-correlating *symbols*.

The communication protocol we are examining for the intercluster level defines at least the following five classes of messages:

- Introduction and synchronization
- General information
- Detection
- Interpretation
- Warnings

Messages of the introduction class are the ones used by clusters to introduce themselves to other clusters, to announce their names, detection capabilities, types, positions, coverages, etc.

Messages of this class also are used for time synchronization between clusters.

Messages of the general information class are used to define targets (i.e., give the association of symbolic target with a detection profile/signature), exchange countermeasures information, etc.

Messages of the detection class are those used to interrogate other clusters for the presence of detection features at some time-space interval. Messages of this type also are used in response to such inquiries.

Messages of the interpretation class are used to discuss symbol features, target presence, and the like. They are used by clusters to solicit supporting evidence for hypotheses and to provide either supporting or contradicting evidence to hypotheses suggested by other clusters.

Messages of the warning class are those used by clusters to warn others of expected events (e.g., "an aircraft of a characteristics X will appear at position P at time T"), and to solicit similar warnings (e.g., "Let me know if within time T you see the event X at position Y", where X is either a target or a certain detection feature).

It is too early at this stage to be able to define the grammar and the format of these messages.

The Distributed Sensors Network communicates with the external world by using the same class of messages.

9. PACKET RADIO TERMINAL MOCKUP

Participating Staff:
Vern Dieter
George Dietrich
Tom Ellis
Oralio Garza
Nelson Lucas
Robert Parker

The ISI Model Shop produced a mockup of a portable terminal as shown in the several views of Fig. 1. This model was constructed to illustrate the approximate size and weight of a minimum-size (maximum portability) radio-coupled terminal, as well as some significant human factors issues. The terminal is assumed to have the following characteristics:

1. All normal alphanumeric computer terminal functions with full ASCII keyboard, display of 25 - 80 character lines, upper and lower case text and inverse video highlighting.

2. 480 x 250 raster point graphics or Fax.

3. Direct tablet over display with stylus graphic input.

4. Digital voice communications.

5. Integral packet radio transceiver.

6. Self-contained, rechargeable batteries for at least 10 hours operation.

Prior to the physical design of the model and its form factor, brief design studies were performed on the three main sections of the unit: display, digital support electronics, and radio unit.

For the display, technologies were studied not on the basis of maturity but the promise of very good viewing characteristics and very low power consumption in a thin flat panel. No established technology yet exists to satisfy these goals, although the (very immature) electrophoretic display technology is very promising if it can be integrated with thin film transistor technology to provide the dot array addressability.

In-depth discussions with several laboratories experimenting with either thin film transistors or electrophoretics provided much encouragement and the essential realistic specifications, including power requirements, size (including integrated addressing/scanning logic on the periphery), thickness, various environmental considerations, interfacing requirements, and practical dot resolution capabilities.

A preliminary detailed circuit design was performed for the thin film x,y matrix and peripheral addressing logic to illuminate developmental issues. Fine-grain test panels (100 lines/inch) of electrophoretic material were also produced, again to illuminate problem areas. These preliminary studies were sufficient to inspire confidence that a concentrated developmental effort will have positive results, providing a significant step in alternatives for low-power display technologies.

Photographic mockups of full displays were produced at ISI, emulating as closely as possible the color and contrast ratios and dot separation/coverage the technology would be expected to produce. From these, a display of 90 dots per inch with active area of about 3 x 6 inches was chosen to maintain 5 x 7 text characters large enough to be easily readable, yet minimize overall screen size.

A logical design for all digital support functions was performed to enough detail to provide a close count for CMOS chip requirements based on the RCA 1802 microprocessor. Also, circuits were designed to handle the higher voltage interfacing requirements to the thin film display panel to identify the component and power requirements in this area.

Volume and power requirements for the radio unit were taken from a previous ARPA-supported design study.

The outside dimensions of the resulting package design are 6-7/8" x 3-7/8" x 1-1/2". As shown, the design includes a separable radio unit to allow flexibility in antenna placement to optimize range capabilities. This mockup was delivered to ARPA-IPTO on May 10, 1977.

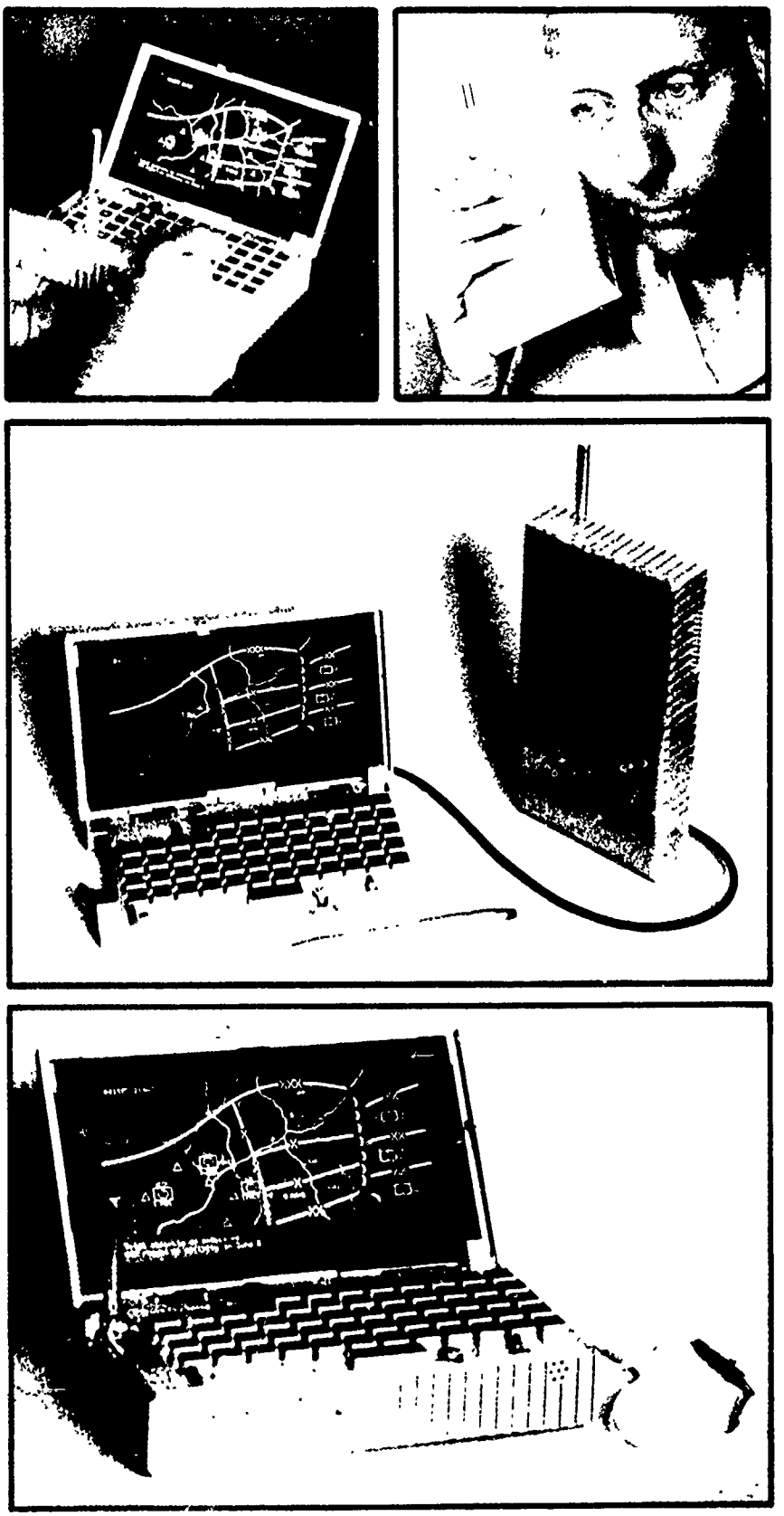


Figure 9.1 Various views of the Packet Radio Terminal mockup

10. USER-DEDICATED RESOURCE

Project Staff: Chloe Holg

BACKGROUND

The ARPA/IPTO research community's continuous expansion of the frontiers of information processing technology is reflected in a constant expansion and modification of informational tools available on the ARPANET. These tools tend to have inadequate documentation (not a researcher's primary concern); moreover, their large numbers and many subsequent modifications makes it difficult even to keep track of them all.

ARPA has been introducing new users to the network at an increasing rate for two major purposes. First, because the network is a superb communications medium, some users are being placed on-line whose main function is to communicate with the ARPA program managers and/or the research community, both as a test of current capabilities and as an impetus to ongoing research. Second, ARPA must transfer the results of its research programs to other agencies, and does so partly by educating selected users in the use of new information processing tools. Both purposes were no doubt involved in the introduction of military users such as the Advanced Command and Control Architectural Testbed groups; the Systems Development Laboratory and the Naval Ocean Systems Center; the Naval Research Laboratory; and Rome Air Development Center.

NEED FOR THE RESOURCE

The full benefits of network usage were not being fully realized for either ARPA or the military services. These important users faced an initial barrier, partly because many were new to on-line computer systems in general, partly because systems turnover was so rapid, and certainly because up-to-date introductory documentation was lacking. (It is hard to consider objectively the merits of a new technology when one has difficulty getting through a TIP and logging in to TENEX.) It's well known that researchers normally do not produce adequate initial usage documentation for nonprogrammers; therefore, the best solution was to make a single individual responsible for helping these new users overcome that initial barrier by keeping track of relevant developments and communicating clearly (i.e., with minimal computer jargon). This user resource was established at ISI in FY77.

Although new users eventually become experienced users, there is no end to the problem: newer users are constantly introduced because of changes in the IPTO program

audience and the normal turnover in military personnel. These new users are contacted as soon as their accounts are installed on the ISI machines; they are interfaced to the available network facilities by means of three levels of appropriate documentation.

Both short-term solutions (answering questions, solving individual problems) and long-term solutions (guiding users in solving common problems, making available better procedures and documentation) are routinely provided. User problems are analyzed as they occur and appropriate action decided upon. Useful inputs are also provided to system programmers to better meet the users' needs.

There are two groups one might think of who are already involved with new users: the TENEX systems groups and single-system advisors. The former provide help to programmers who need detailed information about how to utilize the operating system resources in their programming jobs; they work very well with experienced, jargon-understanding users but do not adequately address the more diverse population of ARPA-related users. The latter can provide help with a particular tool (for example, NLS has its own training staff to conduct classes and answer users' questions, and several other subsystems have mail addresses for comments, queries, or complaints). While these special-purpose tutorial aids are excellent for a stable user of a single system, ARPA's new (and old) user population usually requires many tools produced by many system developers. Thus coordinated information is necessary about what tools are available for specific purposes and where to obtain the more detailed information on how to use them.

The TENEX manuals that existed prior to the introduction of the ISI/TENEX manuals are, in the main, reference manuals (e.g., *TENEX Exec Manual, TENEX User's Guide*); these are not appropriate to introduce new users to the network, since they are both too voluminous and too incomplete to be useful for novices. The introductory ISI/TENEX manuals which now exist, which first appeared in November 1975, must be frequently updated and useful basic tools added or deleted when appropriate [1]. Both the rapid distribution of the manuals and the appearance of new documentation (e.g., *The XED Beginning Instruction Manual* [2] necessitated a second iteration in April 1976 and a third in November 1976. Further, it became necessary to focus attention on the rather impressive set of new users accessing the ARPANET via the TIP for purposes of mail-handling, and a more basic, primer-like manual [3] was published in April 1977.

With ongoing efforts among the system support groups to effect standardization of resources available in TENEX across the ARPANET machines, this task has been and is yet providing broad support to TENEX users across the the entire network. Techniques employed to develop and provide TENEX user documentation and assistance are being applied to the ARPANET TOPS-20 operating system. Currently, information is being accumulated; system folklore is rapidly being translated into system fact, the basis of a TOPS-20 primer. Special attention is being focused on TENEX users being transitioned from TENEX to the TOPS-20 system. Further, plans are under way to provide basic user documentation for the ISI-developed Military Message Experiment (MME).

SUMMARY

The first year of this innovative approach to computer user needs and problems has been productive. The need to provide simple and direct documentation (ARPANET/TENEX/MSG Primer) was recognized and addressed; the existence of a single target for users' questions and problems has been and will continue to be a large factor in providing a positive and useful environment to both new and more experienced users of the ARPANET community.

REFERENCES

1. Holg, Chloe, *The JOY of TENEX.;3..the basics* and *MORE JOY of TENEX.;3...some of the refinements*, Information Sciences Institute, November 1976.

2. *The XED Beginning Instruction Manual*, ISI/TM-76-3, Information Sciences Institute, May 1976.

3. Holg, Chloe, *ARPANET TENEX Primer and MSG Handling Program*, Information Sciences Institute, ISI/TM-77-4, April 1977.

11. ARPANET TENEX SERVICE

Technical Staff:

Marion McKinley, Jr.
Pete Alfvin
Alan E. Algustyniak
Wanda N. Canillas
Dale Chase
Vernon Dieter
George Dietrich
Glen W. Gauthier
Kyle P. Lemons
James M. Lieb
Donald R. Lovelace
Raymond L. Mason
John P. Metzger
William H. Moore
Edward D. Mortenson
Donna Nagel
Robert Parker
Clarence Perkins
Vernon W. Reynolds
Dale S. Russell
Barden E. Smith
Lee P. Taylor
Leo Yamanaka

Support Staff:

Carol Carreon
Larry Fye
Oralio E. Garza
James Griffin
Randolph A. Reidel
Robert M. Robbins
Leroy Ryan
Gary Seaton
Ronnie Simpson
Scott Smith
Michael E. Vilain
Pat Weiber
Deborah C. Williams

INTRODUCTION

The ISI ARPANET TENEX service project consists of two computer centers: a local and a remote installation. The local computer center is operated as a nonclassified developmental and service center in support of a broad set of ARPA requirements, ARPA projects, ARPA contractors, and military users. It currently services more than 1200 directories, some of which are multiplexed by several users. Approximately 95 percent of the users access the facilities via the ARPANET from locations extending from Europe to Hawaii. The remote computer center is operated in a classified environment as part of the Advanced Command and Control Architecture Testbed (ACCAT) at the Naval Ocean Systems Center (NOSC), San Diego, California. It currently services ARPA and Navy contractors who are involved in the joint ARPA and U. S. Navy Command and Control experiment. The classified computer center is presently accessible only by those personnel physically located within the classified facility. Future plans will allow remote users access via a miniaturized classified ARPA network.

The local computer center consists of four large-scale Digital Equipment Corporation (DEC) central processors (one KI-10, one KL-1090T and two KA-10s), Bolt Beranek and Newman (BBN) virtual memory paging boxes, large-capacity memories, on-line swapping and file storage, and associated peripherals (see Figs. 11.1 and 11.2). All of the above-mentioned systems presently run under control of the TENEX (originally developed by BBN) or TOPS-20 operating system, which supports a wide variety of simultaneous interactive users. In addition, the local facility supports other processors, such as several DEC PDP-11/40's, one DEC PDP-11/45, and associated peripheral devices. The remote

center consists of three large-scale DEC central processors (one KL2040T, one KA-10, and one PDP-11/70), virtual memory paging box, large capacity memories, on-line swapping and file storage and associated peripherals (see Fig. 11.3). The KA-10 runs under the TENEX operating system, the KL-2040T runs under the DEC TOPS-20 operating system and the PDP-11/70 runs under Bell Laboratory's UNIX operating system.

HARDWARE

New hardware acquired during the past year for the local computer center includes one complete DEC KL1090T (TOPS-20) system with 1024K of external memory and associated peripheral devices and four CALCOMP CD230 disk spindles, of which two each have been added to TENEX systems ISIA and ISIC, thereby increasing the on-line swapping and file storage capabilities on each of these two systems. An additional 128K of Ampex high-speed memory has been added to system ISIB, thereby increasing the memory capacity of this system to 512K. A new Data Products 2550 line printer has also been attached to the new KL1090T (ISIE) and is now providing ISI with good quality printing service. A DEC communications device (DN-87) inclusive with a PDP-11/40 has been received and is installed on ISIB, and is allowing all in-house users direct access to ISIB. The PDP-11/40 portion of this (DN-87) device also allows direct connection of a GENISCO Graphics Display System to ISIB. The GENISCO was purchased and installed in support of ISI's involvement with graphics within the ACCAT project at NOSC.

Also included within the ISI local computer center are two BBN H-516 Interface Message Processors (IMP), one DEC PDP-11/40 and Xerox Graphics Printer (XGP), one DEC PDP-11/45 with an SPS-41 Signal Processing System and a Floating Point System AP-120B (FPS) (configured as a speech processor), one Microprogrammable Processor (MLP-900), and several associated peripheral devices such as disk, memories, terminals, etc. from various manufacturers and several special ISI-designed and developed interfaces.

Systems ISIA and ISIC are currently designated as priority systems and are therefore cross-connected (cabled) in such a manner so that if one of these systems crashes or is otherwise unavailable because of hardware/software maintenance or development, then the other system may be started as a back-up replacement system and service continued after a brief (15 minutes or less) delay to switch the file storage media and one cable.

ISI has been successful in negotiating a trade of ISIA's 256K memory (which consisted of sixteen large racks of equipment) back to DEC for a new compatible (in capacity and speed) memory contained in only one rack, which reduces power and air conditioning requirement by a factor of approximately twenty for this system. Because of this trade, the ISI local computer center is now able to accommodate at least one and possibly two additional large-scale computer systems. Figure 11.2 shows the current ISI local computer center configuration.

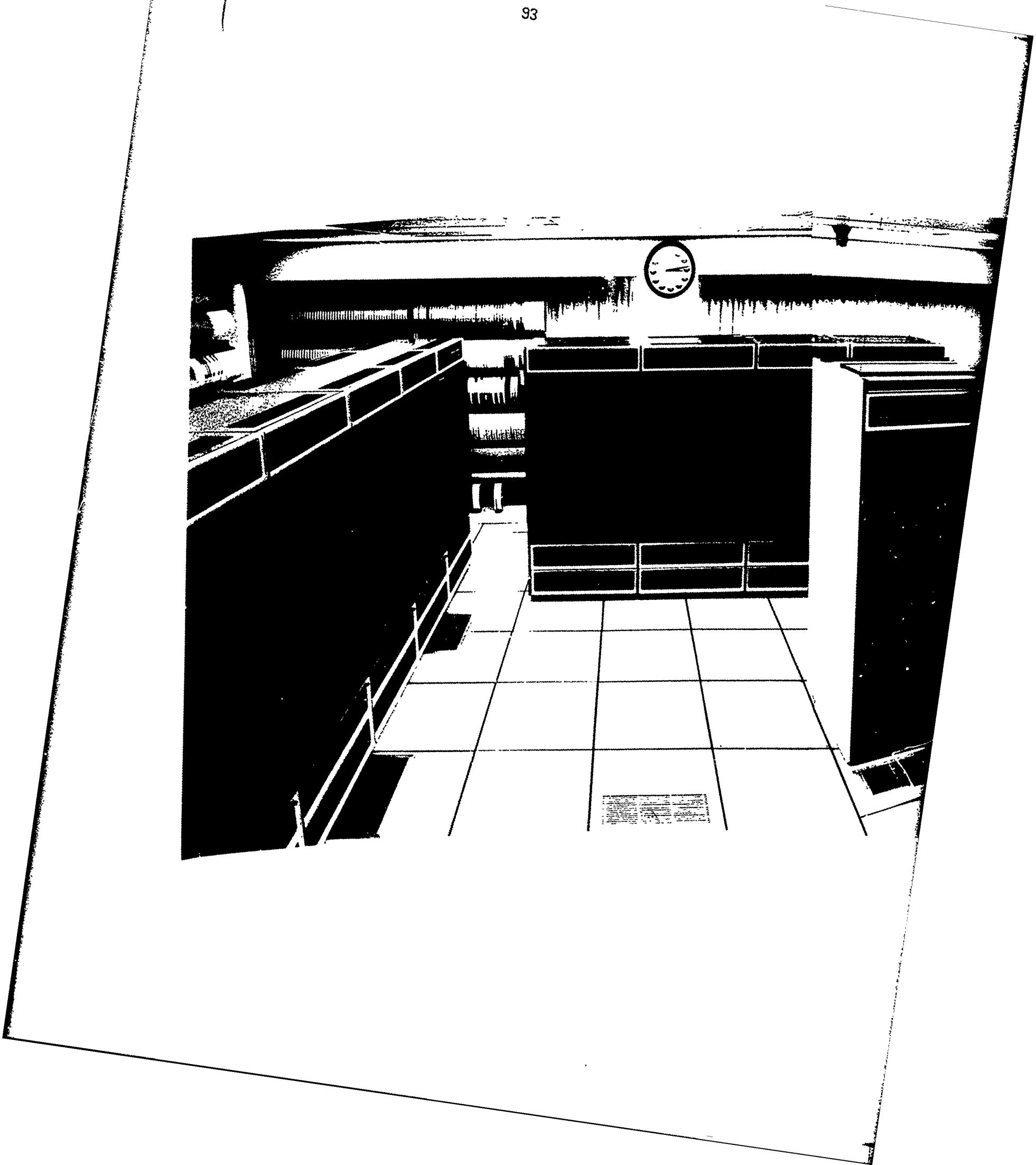
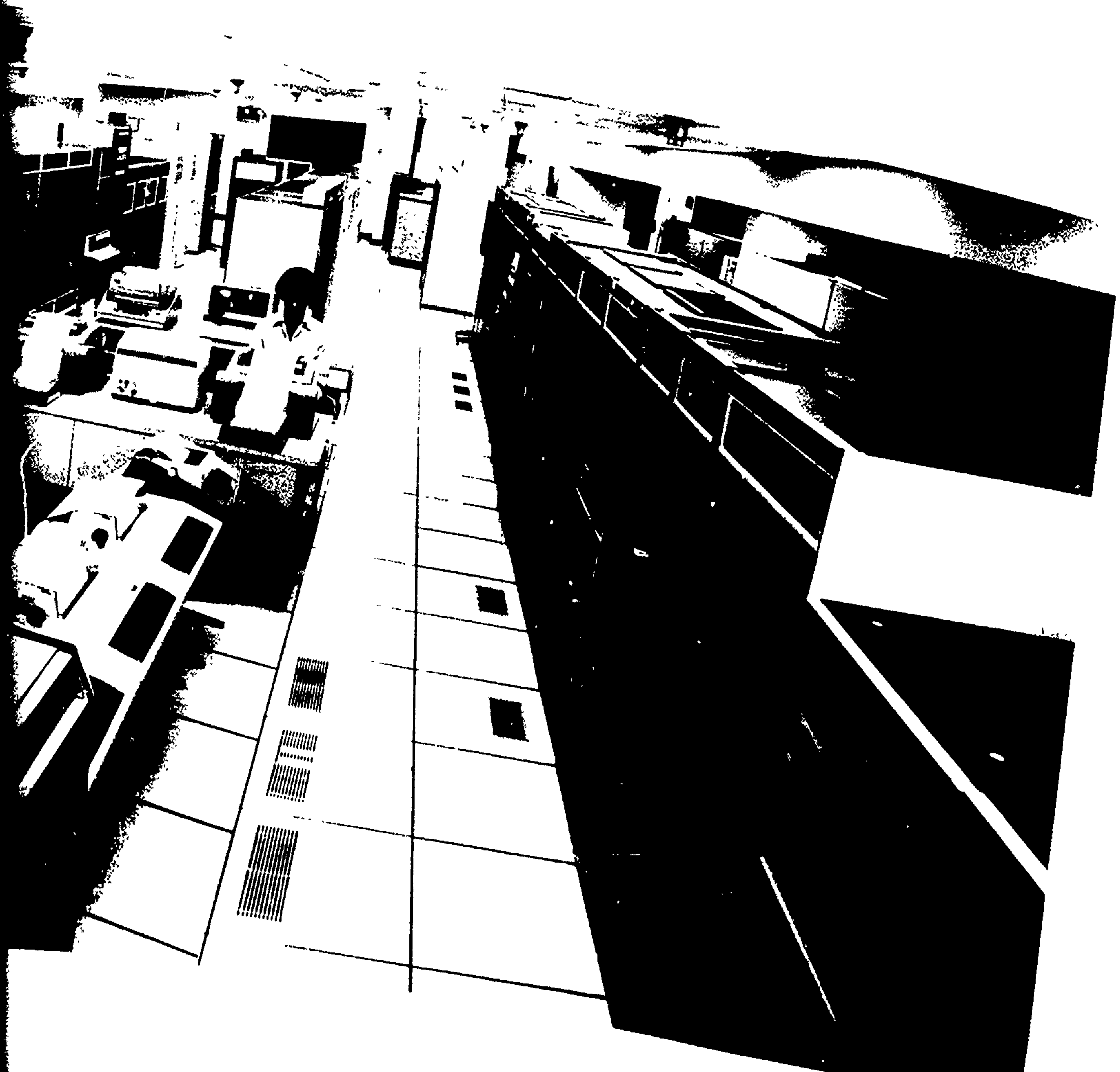


Figure 11.1 Composite photograph of the ISI computer room.



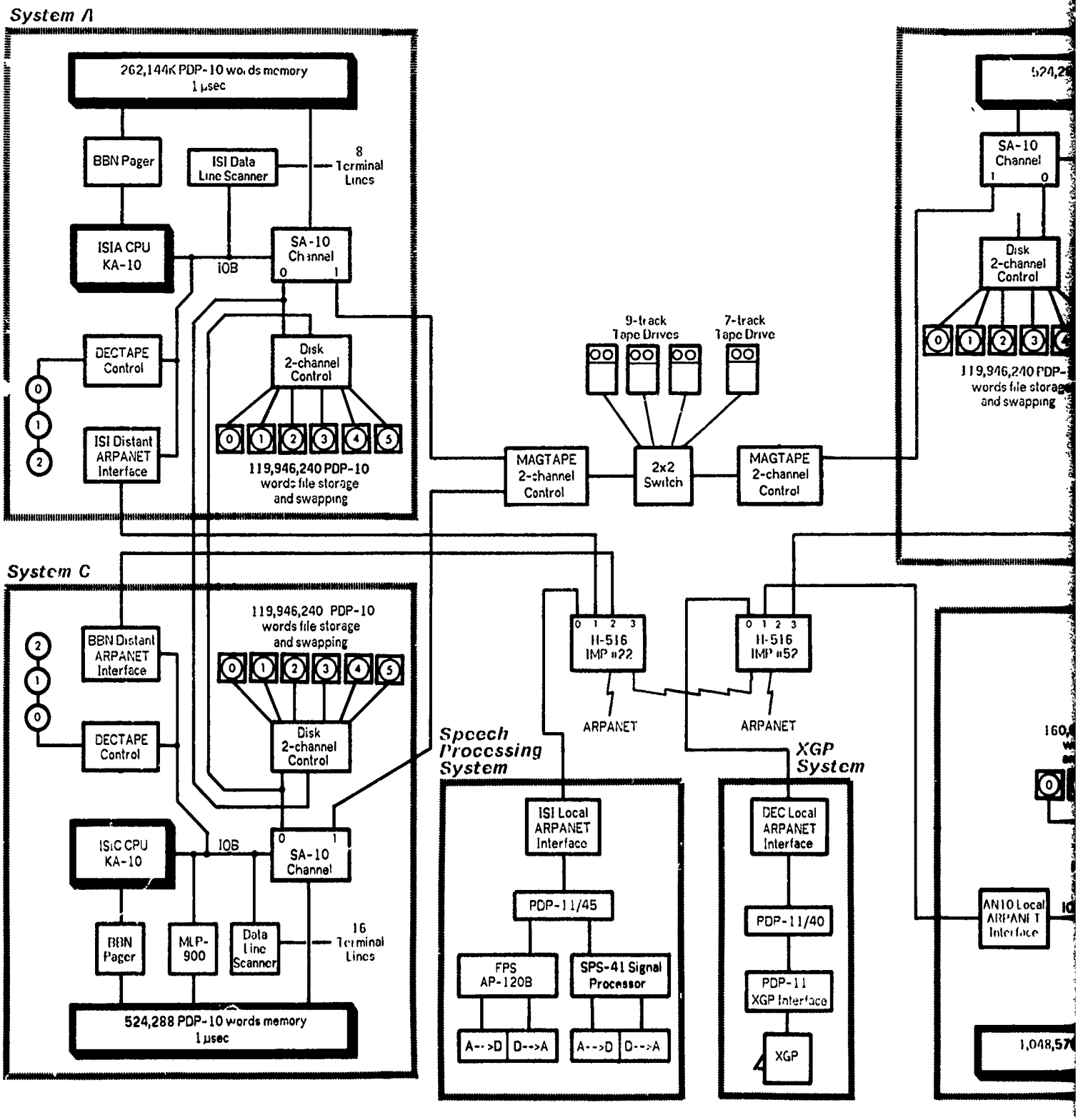
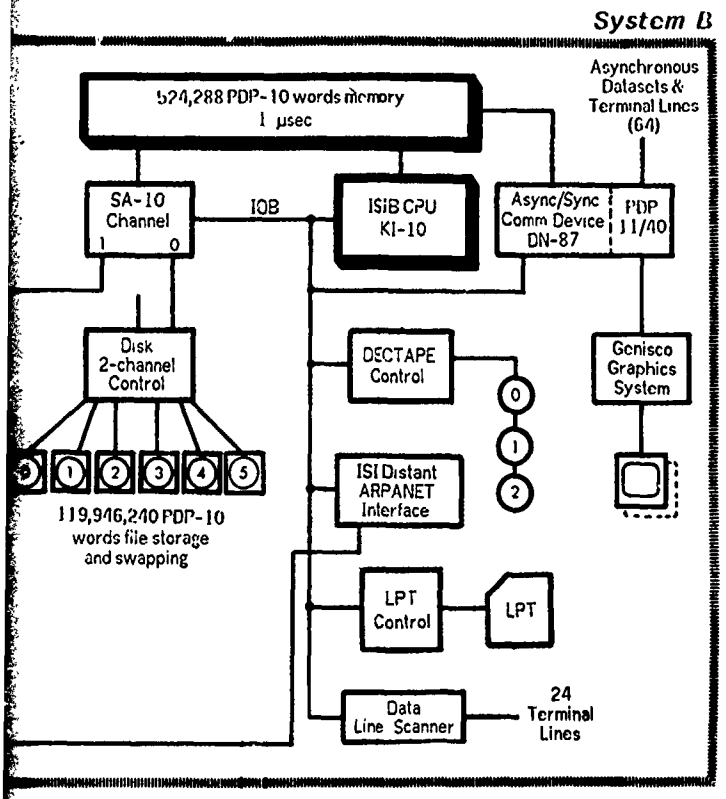
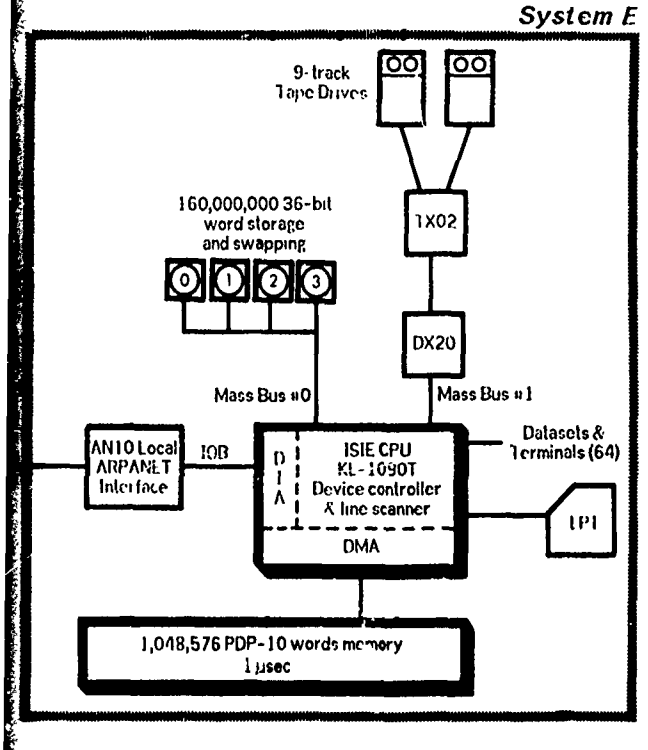
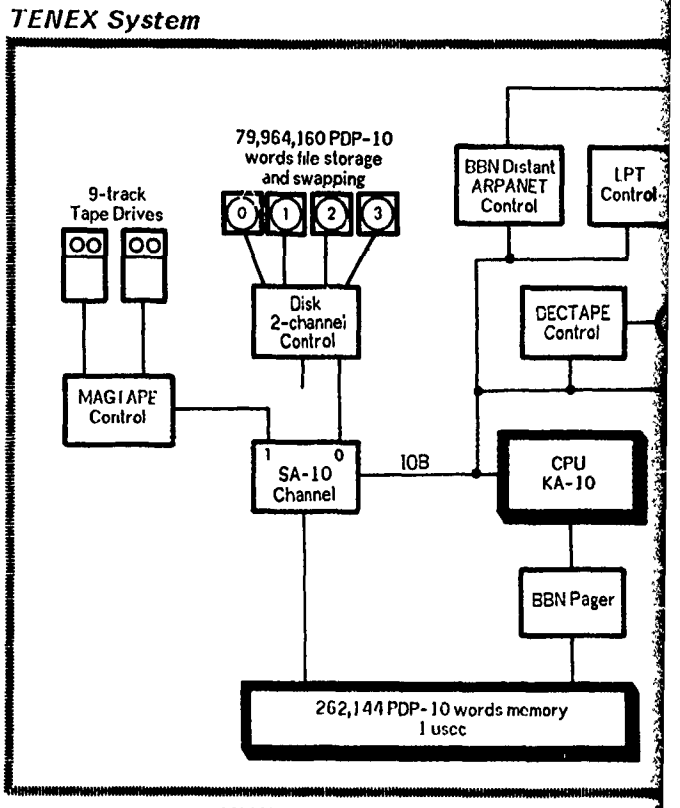
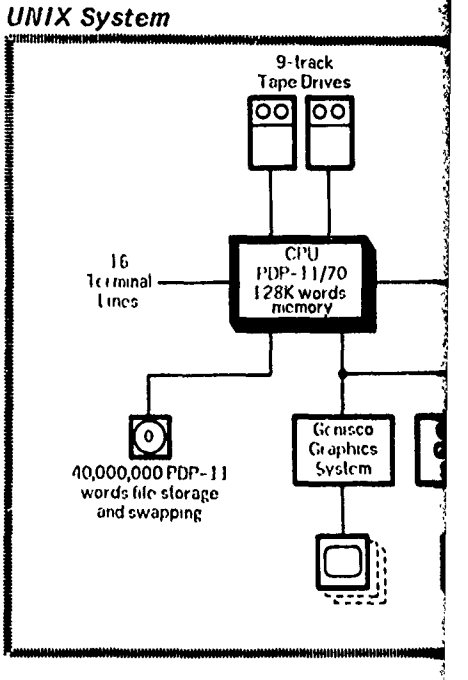


Figure 11.2 Diagram of local ISI ARPANET TENEX service facility.









ice facility.

Figure 11.3 Diagram o

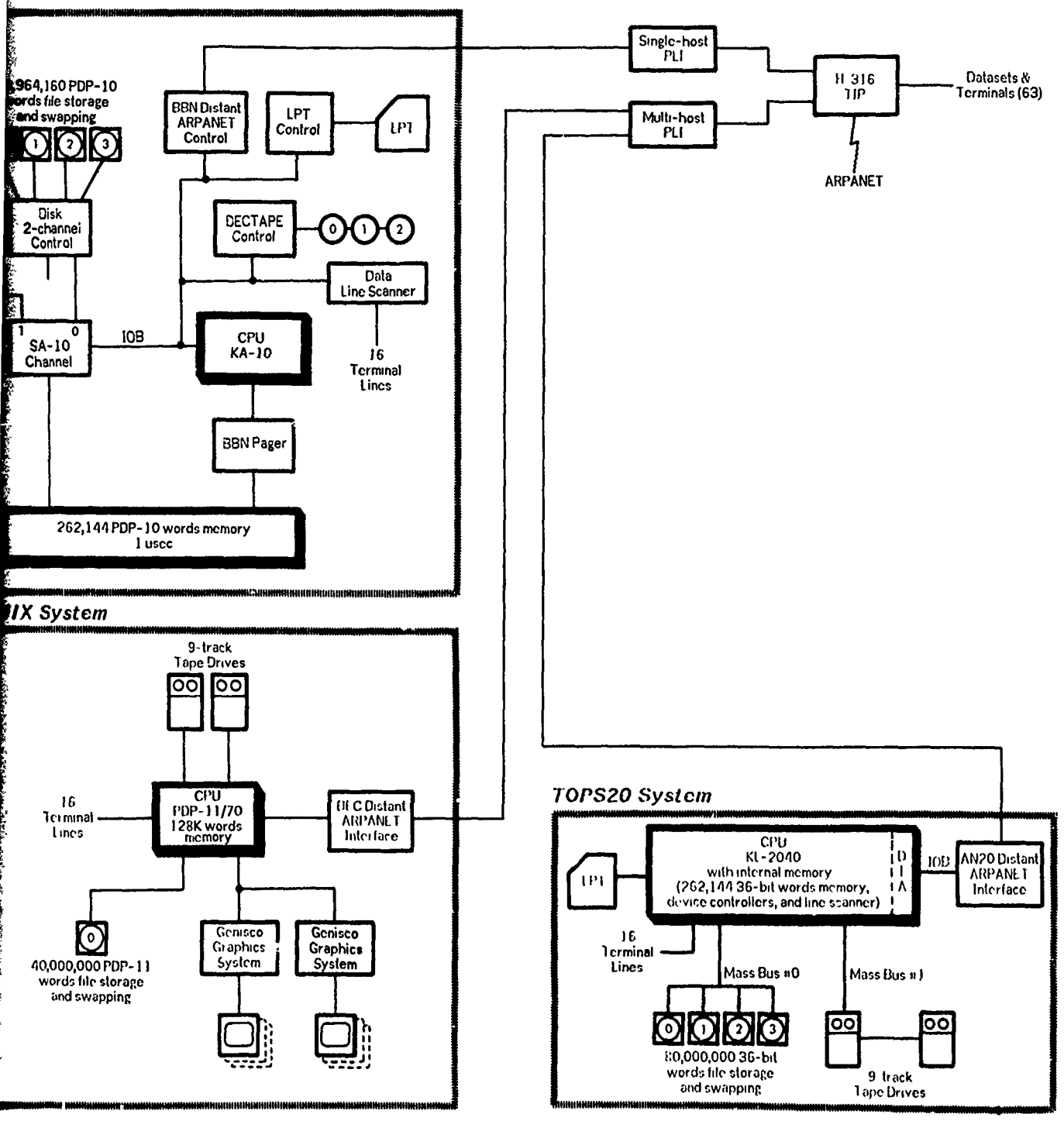


Figure 11.3 Diagram of remote ISI ARPANET TENEX service facility.

The remote computer center at NOSC became operational on December 15, 1976 with a DEC KA-10 (TENEX) system which was previously designated as ISID and housed within the ISI local computer center. New equipment purchased for this system included a CALCOMP 1040 Magnetic Tape Controller and two CALCOMP 347A Magnetic Tape Drives. Also, a Data Products 2550 line printer and DEC Line Printer Controller were added to this system shortly after becoming operational. During April 1977, a complete large-scale DEC PDP-11/70 (UNIX) system, consisting of 128K memory and associated peripheral devices, was installed and became operational. In May of 1977, a complete DEC KL-2040T (TOPS-20) system, previously housed within the ISI local computer center, was installed; it became operational in June. Figure 11.3 shows the current ISI remote computer center configuration.

SOFTWARE

During the year, much of the software expertise within the ISI ARPANET TENEX service project has been devoted to modification of the DEC TOPS-20 operating system so that it would appear to the user community more compatible with the TENEX operating system. Although the DEC TOPS-20 operating system is a direct descendant of the TENEX operating system, and is quite similar in its operation, there are many differences as well. These differences include expanded functions of existing TENEX JSYS calls, new JSYS calls implementing features which are standard in the DEC TOPS-20 operating system, and unfortunately conflicting changes to existing TENEX JSYS calls which required reprogramming efforts in standard subsystem software and many user programs. The DEC TOPS-20 operating system was also void of many crucial features found in the TENEX operating system which are required by our wide variety of users, especially the National Software Works (NSW) efforts.

The first major addition to the TOPS-20 system was the Network Control Protocol (NCP), which, with the standard network utility programs (TELNET, FTP, RESEXEC), made the TOPS-20 system a useful tool to users of the ARPA network. The next major addition was the Create Job (CRJOB) JSYS to allow the creation of new jobs under program control. The most recent addition was the JSYS Trap Facility, which allows superior forks to monitor and control JSYS calls from inferior forks. Numerous software problems (bugs) were found and repaired in various areas such as scheduling, page management, I/O transfer and several JSYS calls. All changes introduced into the TOPS-20 operating, system were necessary for the running of the large volumes of ARPA network user's software. All changes were coded following DEC's standard coding conventions and were reported to DEC with the hopes that these changes would become part of their official TOPS-20 system releases, and thus become maintainable by DEC. Many of the subsystems available on the TENEX system were required by ARPA network users if they were to effectively use the TOPS-20 system in a manner similar to TENEX. ISI decided to make all required changes in such a manner so that the same code (Core Image) would run under either operating system, making any necessary system-dependent selections at run time

rather than assembly or load time. As of this date, most of them have been reprogrammed and are operational under the TOPS-20 operating system.

To help reduce the high loads on our existing TENEX systems and provide a more generally useful service, a BATCH facility has been organized and installed. With this service, users will be able to submit their compute-bound tasks to background jobs which will allow these tasks to run during non-prime and low load conditions, leaving their terminals free for interactive use. The number of background jobs will be kept few to reduce the immediate demand for CPU and memory cycles. Thus, interactive users will see better system response time and the compute-bound jobs will require less system time because of reduced system overhead.

It became obvious last year that one of the most critical support functions that ISI should provide across all of our systems was a consistent level of subsystems (e.g., editors, compilers, assemblers, and utilities). This became a very tedious and time-consuming manual task. This year, to automate this procedure a File Update Support System (FUSS) was developed which makes use of the existing message and file transfer protocols to announce and subsequently retrieve new updated subsystems. The basic cycle is as follows: A system staff member updates a subsystem at a cooperating site, then puts instructions into a file describing which cooperating sites are to be notified, which files were changed, and any announcement(s) the users should see regarding the update. At each cooperating site, there is a background program which, at prearranged specified times, will look into the systems staff directories for these instruction files, assemble one message for each site involved, giving the file retrieval and deletion instructions and update announcements, then cause MAILER to transmit them. Later, this same program will read its own mailbox for instructions on what it is supposed to do. Upon completion of the file shuffling, one message with all of the announcements is composed and sent to the mailbox which is shown to users upon log-in.

FUSS has been used successfully among all of the ISI TENEX and TOPS-20 systems, and has been adopted by BBN and Stanford Research Institute to keep their systems in sync. Further development of this system is continuing so that it will be easier to use and less prone to typing mistakes (e.g., file specifications, etc.). Eventually, it will be capable of keeping cooperative sites up-to-date with the latest versions of all subsystems with minimal input.

All of the ISI TENEX systems are running the latest released version of TENEX (V134) and have been fairly stable with just a few minor bug fixes installed and a few new utility JSYS's installed. Recently, the TENEX systems were modified to read the shareable SAVE files produced by the TOPS-20 systems, which will ease the transferability of software between TOPS-20 and TENEX systems.

SUPPORT PERSONNEL

ISI provides seven-day-a-week, twenty-four-hour-a-day operator, software, and hardware support for the local computer center. At least one operator is physically on-site at all times, and the systems programmers and computer service engineers either are physically on-site or are scheduled for one-hour on-call service. The ISI remote computer center is currently manned as a five-day-a-week, eight-hour-a-day type of operation and all support personnel are physically on-site only during these times. It is expected that on or about April 1, 1978, ISI will commence seven-day-a-week, twenty-four-hour-a-day operator coverage for the systems within this computer center. This additional coverage will be necessary to accommodate remote and in-house users who are expected to require access to these systems at all hours.

RELIABILITY

To provide required hardware/software preventive and/or corrective maintenance of the equipment, ISI will continue scheduling each of the TENEX/TOPS-20 systems as "out of service" (unavailable to users) for seven contiguous hours each week. The remaining 161 hours of each week are intended to be devoted entirely (100 percent) to user service. The actual long-term up-time for the network service machines has again exceeded 99 percent of scheduled up-time for the last year.

LOCAL PROJECT SUPPORT

The local service center has been used extensively in support of local projects. The ISI staff makes use of the available standard subsystems and some staff members have written subsystems and utilities to support their own projects. The facility also supports less frequently used subsystems at the special request of users (e.g., PDP-11 cross-assemblers and the DECUS Scientific Subroutine Package).

INSTITUTE PUBLICATIONS

Research Reports

Abbott, Russell J., *A Command Language Processor for Flexible Interface Design*, ISI/RR-74-24, February 1975.

Anderson, Robert H., *Programmable Automation: The Future of Computers in Manufacturing*, ISI/RR-73-2, March 1973; also appeared in *Datamation*, Vol. 18, No. 12, December 1972, pp. 46-52.

---, and Nake M. Kamrany, *Advanced Computer-Based Manufacturing Systems for Defense Needs*, ISI/RR-73-10, September 1973.

Balzer, Robert M., *Automatic Programming*, ISI/RR-73-1 (draft only).

---, *Human Use of World Knowledge*, ISI/RR-73-7, March 1974.

---, *Imprecise Program Specification*, ISI/RR-75-36, May 1976; also appeared in *Calcolo*, Vol. XII, Supplement 1, 1975.

---, *Language-Independent Programmer's Interface*, ISI/RR-73-15, March 1974; also appeared in *AFIPS Conference Proceedings*, Vol. 43, AFIPS Press, Montvale, N. J., 1974.

---, Norton R. Greenfeld, Martin J. Kay, William C. Mann, Walter R. Ryder, David Wilczynski, and Albert L. Zobrist, *Domain-Independent Automatic Programming*, ISI/RR-73-14, March 1974; also appeared in *Proceedings of the International Federation of Information Processing Congress*, 1974.

Bisbey, Richard L., Jim Carlstedt, Dale M. Chase, and Dennis Hollingworth, *Data Dependency Analysis*, ISI/RR-76-45, February 1976.

---, and Gerald J. Popek, *Encapsulation: An Approach to Operating System Security*, ISI/RR-73-17, December 1973.

Britt, Benjamin, Alvin Cooperband, Louis Gallenson, and Joel Goldberg, *PRIM System: Overview*, ISI/RR-77-58, March 1977.

Carlisle, James H., *A Tutorial for Use of the TENEX Electronic Notebook-Conference (TEN-C) System on the ARPANET*, ISI/RR-75-38, September 1975.

Carlstedt, Jim, Richard L. Bisbey II, and Gerald J. Popek, *Pattern-Directed Protection Evaluation*, ISI/RR-75-31, June 1975.

Cohen, Dan, *Specification for the Network Voice Protocol*, ISI/RR-75-39, March 1976.

Ellis, Thomas O., Louis Gallenson, John F. Heafner, and John T. Melvin, *A Plan for Consolidation and Automation of Military Telecommunications on Oahu*, ISI/RR-73-12, June 1973.

Gallenson, Louis, *An Approach to Providing a User Interface for Military Computer-Aided Instruction in 1980*, ISI/RR-75-43, December 1975.

Good, Donald I., Ralph L. London, and W. W. Bledsoe, *An Interactive Program Verification System*, ISI/RR-74-22, November 1974; also appeared in *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 1, March 1975, pp. 59-67.

Guttag, John V., Ellis Horowitz, and David R. Musser, *The Design of Data Type Specifications*, ISI/RR-76-49, November 1976.

Heafner, John F., *A Methodology for Selecting and Refining Man-Computer Languages to Improve Users' Performance*, ISI/RR-74-21, September 1974.

---, *Protocol Analysis of Man-Computer Languages: Design and Preliminary Findings*, ISI/RR-75-34, July 1975.

Igarashi, Shigeru, Ralph L. London, and David C. Luckham, *Automatic Program Verification I: A Logical Basis and Its Implementation*, ISI/RR-73-11, May 1973; also appeared in *Artificial Intelligence Memo 200*, Stanford University, May 1973 and *Acta Informatica*, Vol. 4, No. 2, 1975, pp. 145-182.

Kamrany, Nake M., *A Preliminary Analysis of the Economic Impact of Programmable Automation Upon Discrete Manufacturing Products*, ISI/RR-73-4, October 1973.

Kimbleton, Stephen R., *A Heuristic Approach to Computer Systems Performance Improvement. I: A Fast Performance Prediction Tool*, ISI/RR-74-20, March 1975.

Levin, James A., and James A. Moore, *Dialogue Games: Meta-Communication Structures for Natural Language Interaction*, ISI/RR-77-53, January 1977.

London, Ralph L., Mary Shaw, and William A. Wulf, *Abstraction and Verification in ALPHARD: A Symbol Tale Example*, ISI/RR-76-51, January 1977.

Mann, William C., *Dialogue-Based Research in Man-Machine Communication,* ISI/RR-75-41, November 1975.

---, *Why Things Are So Bad for the Computer Naive User,* ISI/RR-75-32, March 1975.

---, James A. Moore, James A. Levin, and James H. Carlisle, *Observation Methods for Human Dialogue,* ISI/RR-75-33, July 1975.

---, James H. Carlisle, James A. Moore, and James A. Levin, *An Assessment of Reliability of Dialogue Annotation Instructions,* ISI/RR-77-54, January 1977.

---, *Man-Machine Communication Research Final Report,* ISI/RR-77-57, February 1977.

Martin, Thomas H., Monty C. Stanford, F. Roy Carlson, and William C. Mann, *A Policy Assessment of Priorities and Functional Needs for the Military Computer-Aided Instruction Terminal,* ISI/RR-75-44, December 1975.

Miller, Lawrence H., *An Investigation of the Effects of Output Variability and Output Bandwidth on User Performance in an Interactive Computer System,* ISI/RR-76-50, December 1976.

Moore, James A., James A. Levin, and William C. Mann, *A Goal-Oriented Model of Natural Language Interaction,* ISI/RR-77-52, January 1977.

Oestreicher, Donald R., *A Microprogramming Language for the MLP-900,* ISI/RR-73-8, June 1973; also appeared in the Proceedings of the ACM Sigplan Sigmicro Interface Meeting, New York, May 30-June 1, 1973.

Richardson, Leroy, *PRIM Overview,* ISI/RR-74-19, February 1974.

Rothenberg, Jeff, *An Editor to Support Military Message Processing Personnel,* ISI/RR-74-27, June 1975.

---, *An Intelligent Tutor: On-Line Documentation and Help for A Military Message Service,* ISI/RR-74-26, May 1975.

Shaw, Mary, William A. Wulf, and Ralph L. London, *Abstraction and Verification in ALPHARD: Iteration and Generators,* ISI/RR-76-47, September 1976.

Tugender, Ronald, and Donald R. Oestreicher, *Basic Functional Capabilities for a Military Message Processing Service,* ISI/RR-74-23, May 1975.

Wilczynski, David, *A Process Elaboration Formalism for Writing and Analyzing Programs,* ISI/RR-75-35, October 1975.

Wulf, William A., Ralph L. London, and Mary Shaw, *Abstraction and Verification in ALPHARD: Introduction to Language and Methodology*, ISI/RR-76-46, July 1976.

Yonke, Martin D., *A Knowledgeable, Language-Independent System for Program Construction and Modification*, ISI/RR-75-42, December 1975.

Special Reports

Annual Technical Report, May 1972 - May 1973, ISI/SR-73-1, September 1973.

A Research Program in the Field of Computer Technology, Annual Technical Report, May 1973 - May 1974, ISI/SR-74-2, July 1974.

A Research Program in Computer Technology, Annual Technical Report, May 1974 - June 1975, ISI/SR-75-3, September 1975.

Bisbey, Richard L., Gerald Popek, and Jim Carlstedt, *Protection Errors in Operating Systems: Inconsistency of a Single Data Value Over Time*, ISI/SR-75-4, January 1976.

Carlstedt, Jim, *Protection Errors in Operating Systems: Validation of Critical Variables*, ISI/SR-75-5, May 1976.

A Research Program in Computer Technology, Annual Technical Report, July 1975 - June 1976, ISI/SR-76-6, July 1976.

Hollingworth, Dennis, and Richard L. Bisbey II, *Protection Errors in Operating Systems: Allocation/Deallocation Residuals*, ISI/SR-76-7, June 1976.

Technical Manuals

Gallenson, Louis, Joel Goldberg, Ray Mason, Donald Oestreicher, and Leroy Richardson, *PRIM User's Manual*, ISI/TM-75-1, May 1975.

XED User's Manual: Beginning Instruction, ISI/TM-76-3, May 1976.

Holg, Chloe, *ARPANET/TENEX Primer and MSG Handling Program*, ISI/TM-77-4, April 1977

RESEARCH AND ADMINISTRATIVE SUPPORT

Institute Administration:
Robert H. Blechen
Judy Gustafson
Lisa Moses
Georgene Petri
Nancy Rhinehart
Kathie Richardson
Pam Wilson

Librarian:
Rose Kattlove

Publications Group:
Nancy Bryan
G. Nelson Lucas

Secretaries to Directors:
Jeannette Christensen
Patricia A. Craig
Arva Morgan