ISI/SR-80-17

1979
Annual Technical Report

October 1978 - September 1979

A Research Program in Computer Technology

Prepared for the

Defense Advanced Research Projects Agency

UNIVERSITY OF SOUTHERN CALIFORNIA *INFORMATION SCIENCES INSTITUTE*
4676 Admiralty Way Marina del Rey California 90291

1979 Annual Technical Report

A Research Program in Computer Technology

ISI/SR-80-17

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: ISI/SR-80-17	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): 1979 Annual Technical Report: A Research Program in Computer Technology	5. TYPE OF REPORT & PERIOD COVERED: Annual Technical Report October 1978-September 1979
6. PERFORMING ORG. REPORT NUMBER	
7. AUTHOR(s): ISI Research Staff	8. CONTRACT OR GRANT NUMBER(s): DAHC 15 72 C 0308
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
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Advanced Research Projects Agency, 1400 Wilson Blvd., Arlington, VA 22209	12. REPORT DATE: June 1979
	13. NUMBER OF PAGES: 192
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	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 is 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, Affirm, Alphard, Euclid, interactive theorem proving, Pascal, program correctness, program verification, rewrite rules, software specification, verification condition.

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

This report summarizes the research performed by USC/Information Sciences Institute from October 1, 1978, to September 30, 1979. The research is aimed at applying computer science and technology to areas of high DoD/military impact.

The ISI program consists of twelve research areas: Program Verification - logical proof of program validity; Military Message Experiment - development

19. KEYWORDS (continued)

2: computer terminals, interactive message service, Military Message Experiment, nonprofessional computer users, SIGMA, terminal-based message service.
3: command and control graphics, computer graphics, high-level graphics language, on-line map display.
4: abstract programming, domain-independent interactive system, HEARSAY III, natural language, nonprocedural language, nonprofessional computer users, problem solving, problem specification, process information.
5: computer network, digital voice communication, network conferencing, packet-switched networks, secure voice transmission, signal processing, speech processing, vocoding.
6: computer mail, gateways, interconnection, internetwork protocol, networks, protocol design, protocols, protocol verification, transmission control protocol, type-of-service.
7: communication protocols, cooperation between decentralized processes, DSN modeling, packet radio network, position-location, rigidity, sensor networks.
8: Ada, Autopsy, CMS-2, DOD-1, program conversion, program equivalence, program translation, source-to-source transformation.
9: command and control, digital voice communication, graphic input device for terminal, multimedia communications, portable terminal, radio-coupled terminal.
10: document preparation, editor/formatter, Network Virtual Terminal, partitioning, state-of-the-art terminal, virtual document.
11: ARPANET, naive computer users, TENEX, user assistance, user documentation.
12: ARPANET, interface, computer network, FPS AP-120B, KA/KI, KL1090T, KL2040, PDP-11/45, resource allocation, TENEX, timesharing, TOPS-20.

20. ABSTRACT (continued)

of an experimental user-oriented message service for potential large-scale military use; Command and Control Graphics - development of a device-independent graphic system and graphics-oriented command and control applications programs; Specification Acquisition From Experts - 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; Internetwork Concepts - exploring aspects of protocols for the interconnection of computer communication networks, specifically the design and prototype implementation of an internetwork computer message system and the design of internetwork host and gateway protocols; Distributed Sensor Networks - developing algorithms and communication protocols to support the operation of geographically distributed sensors; Autopsy - research on source-to-source program translation combining automatic techniques with an interactive system to provide the human manager complete control over the translation process; Packet Radio Terminal System Evaluation - work intended to result in a demonstration-level portable terminal to test and evaluate various solutions to the issues raised by extreme portability in the packet radio environment; MAST - development of a terminal that will support multiprocess or NSW tool interaction through multiple windows and an efficient scope editor that will divide editing responsibility between the host and the terminal; 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.

ISI/SR-80-17

1979
Annual Technical Report

October 1978 - September 1979

A Research Program in Computer Technology

Prepared for the

Defense Advanced Research Projects Agency

Effective date of contract:	**17 May 1972**
Contract expiration date:	**30 September 1981**

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.

UNIVERSITY OF SOUTHERN CALIFORNIA *INFORMATION SCIENCES INSTITUTE*

4676 Admiralty Way Marina del Rey California 90291

RESEARCH AND ADMINISTRATIVE SUPPORT

Institute Administration:
Robert Blechen
Barbara Breitman
Victor Brown
Judy Gustafson
Gina Maschmeier
Patricia Sefton

Graphics Coordinator:
G. Nelson Lucas

Librarian:
David Van de Streek

Publications Group:
Nancy Bryan
Jim Melancon

Secretaries to Directors:
Patricia A. Craig
Arva Morgan

CONTENTS

Summary *iv*

Executive Overview *v*

1. Program Verification *1*

2. Military Message Experiment *17*

3. Command and Control Graphics *35*

4. Specification Acquisition From Experts *43*

5. Network Secure Communication *53*

6. Internetwork Concepts *71*

7. Distributed Sensor Networks *99*

8. Autopsy *117*

9. Packet Radio Terminal System Evaluation *141*

10. Multiapplication Support Terminal *153*

11. User-Dedicated Resource *159*

12. ARPANET TENEX Service *163*

ISI Publications *173*

SUMMARY

This report summarizes the research performed by USC/Information Sciences Institute from October 1, 1978, to September 30, 1979. The research is aimed at applying computer science and technology to areas of high DoD/military impact.

The ISI program consists of twelve research areas: *Program Verification* - logical proof of program validity; *Military Message Experiment* - development of an experimental user-oriented message service for potential large-scale military use; *Command and Control Graphics* - development of a device-independent graphic system and graphics-oriented command and control applications programs; *Specification Acquisition From Experts* - 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; *Internetwork Concepts* - exploring aspects of protocols for the interconnection of computer communication networks, specifically the design and prototype implementation of an internetwork computer message system and the design of internetwork host and gateway protocols; *Distributed Sensor Networks* - developing algorithms and communication protocols to support the operation of geographically distributed sensors; *Autopsy* - research on source-to-source program translation combining automatic techniques with an interactive system to provide the human manager complete control over the translation process; *Packet Radio Terminal System Evaluation* - work intended to result in a demonstration-level portable terminal to test and evaluate various solutions to the issues raised by extreme portability in the packet radio environment; *MAST* - development of a terminal that will support multiprocess or NSW tool interaction through multiple windows and an efficient scope editor that will divide editing responsibility between the host and the terminal; *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 University of Southern California Information Sciences Institute (ISI) is a large information processing research center located in Marina del Rey, California.

ISI's principal focus is research in the field of information processing and digital communication. A majority of the research is application and systems oriented. ISI maintains a strong basic research program to support the application and systems focus. The Institute also is committed to a support role providing both general purpose and special purpose computing to a very large number of external users, as well as supplying most of ISI's internal needs.

The research programs at ISI are summarized below. Although a few of the projects are discrete in nature, most form parts of a larger theme.

For example, the Program Verification, Specification Acquisition, and Autopsy projects should be considered as individual parts of an overall research effort in programming methodology and quality software; the Military Message Experiment, Network Secure Communication, Distributed Sensor Networks, Internetwork Concepts, Command and Control Graphics, Packet Radio Terminal, and MAST are linked elements of a major investigation into man-machine and network communications technology. ARPANET TENEX Service and User-Dedicated Resource are projects whose major role is service to ARPANET users. This mutual reinforcement among the various projects at ISI contributes largely to the productivity of the Institute's research activities.

Program Verification. Program verification concerns the process of constructing computer programs in ways that make it possible to verify that each program meets its specification. Given some description of a programming task, it is ultimately necessary to construct two things: a formal specification of the task and a program for carrying out the task. The central ingredient of the work is the ability to verify the program, i.e., demonstrate by a mathematical proof that the specifications and the program are consistent with each other. The style of specifications (axiomatic definitions plus derived properties of these definitions) and the methods of program construction are all important influences in making verification feasible. Emphasis is on the construction of tools to aid verification, specifically (1) tools to translate programs into conjectures

called *verification conditions*; (2) tools sufficiently powerful and natural to be guided by humans in proof attempts; and (3) tools to express and manipulate formal specifications.

Military Message Experiment. The Military Message Experiment (MME) is a joint program funded by DARPA and the Navy to study the nature of automated interactive message processing in an operational military environment. The test, which ended October 1, 1979, was conducted at Camp Smith, Oahu, Hawaii, by a selected subset of users from Commander in Chief Pacific (CINCPAC) Operations Directorate (J3), which included the Command Center and J3 support groups. The focus of the experiment has been the SIGMA message system, developed at ISI, which runs on a PDP-10 under the TENEX operating system and connects to the AUTODIN network through the Local Digital Message Exchange computer (LDMX). The test environment includes 25 CRT terminals plus seven electrostatic printers, distributed in the Command Center and the offices of the J3 staff. The purpose of the test is to evaluate the utility of an interactive message service and to study its impact on the user organization; in addition, the test provides insight into how future message systems for this type of environment should be designed and built.

Command and Control Graphics. A key requirement for effective command and control is the accurate and timely exchange and effective presentation of relevant information at all levels of command. The application of advanced information processing techniques makes it easier to manage and augment this information exchange, permitting military personnel to better interpret and respond to ordinary as well as crisis situations. In particular, graphic communication of two-dimensional information, such as maps and charts overlayed with relevant tactical and strategic data, can greatly enhance both the quality and the efficiency of communications and provide added depth to the information exchange. The military has employed computer graphics in a variety of special-purpose applications. However, the utility of graphics as a communications vehicle suggests that graphics be made available in a more generalized way, much as voice communication is made available via the AUTOVON system. This has become both attractive and economically feasible with the advent of digital computer communications networks such as the ARPANET and its planned military counterpart, AUTODIN-II, now under construction. To explore this need, ARPA has contracted with ISI to design and develop a graphics system that could exploit the distributed processing capability intrinsic to a computer

network, adapt to the changing communication and processing resources available during and between crisis situations, and evolve to meet future command and control processing and display requirements. In addition, ISI is developing representative application software that uses the graphics system to demonstrate the utility of a distributable, display-device independent graphics capability for command and control applications.

Specification Acquisition From Experts. For several years the SAFE project has been exploring a unique approach to simplifying software specification. Rather than developing a "better" specification language, we have developed a prototype interface between a software specifier and the formal specification to be created. This interface embodies our knowledge of what constitutes good formal software specifications, and it attempts to restate the user's *informal* software specification in the required formalism. We believe such an interface is necessary because formal mechanisms are foreign to man's nature. Only with considerable pain, difficulty, and expense can people be taught to use such formations, but they always revert to informal structures wherever possible. Even mathematical proofs are only partially formalized arguments that a line of reasoning is correct. Formal software specifications must be produced, but that doesn't imply that people should create them. Rather, we believe, people should continue to write informal specifications (such as existing B-5 military software specifications) and use a computer-based tool to produce the formal software specifications. Such a division of effort between man and machine seems much more appropriate, with the computer system responsible for reformulating the user's informal specification into the particular form required, maintaining consistency, avoiding ambiguity, and ensuring completeness. This basically standard language translation task is not difficult if the difference between the input and output languages is small and the input is understood. We have specifically designed our formal specification language to minimize the differences between it and the informal input to eliminate the first of these problems. Thus, the key issue is understanding the input.

Network Secure Communication. As more and more sophisticated technology for command and control is transferred out of the laboratory and into the field, the accompanying need for communications, particularly secure communications, becomes more severe. Voice, graphics, facsimile, and data must be transmitted quickly and

securely. The ISI NSC group has been working to develop, implement, and test systems and methods for packet voice communications, and is beginning work on packet transmission of integrated multimedia communications. The results of this effort can readily be applied to meet the military's needs. The ARPA NSC effort has achieved its first goal, which was to demonstrate a high-quality, reliable packet speech transmission system. This was done using relatively expensive minicomputers as network hosts and high-speed general-purpose signal processors for the necessary speech processing. Present efforts have concentrated on decreasing the size and cost of packet voice facilities and improving the transmitted voice quality. Other problem areas include development of packet voice systems that use wideband networks to handle hundreds or thousands of users and the extension of packet voice techniques into the Internet environment. Both these areas need to be explored in order to build packet voice systems that will serve the large-scale communications needs and the diverse network environments the military is apt to experience in the future.

Internetwork Concepts. This project explores the design and analysis of computer-to-computer communication protocols in multinetwork systems. The project has four task areas: (1) Analysis, (2) Applications, (3) Design, and (4) Concepts. Protocol Analysis is concerned with the correctness of protocols, in particular Transmission Control Protocol. Protocol Applications is concerned with the development of demonstration internetwork applications, in particular a prototype computer message system. Protocol Design is concerned with the development of network and transport protocols, in particular the Internet Protocol and Transmission Control Protocol. Protocol Concepts seeks new approaches in applying packet switching to communication problems.

Distributed Sensor Networks. Military and commercial organizations are looking forward to large integrated data processing systems in the 1980s. Such systems must bind together many separate applications so that information processing techniques such as correlation and synthesis are available across many domains. Ideally, the set of applications would be housed at a central location and users would gain access to this site through conventional network techniques. Several factors and trends--processor costs, communications costs, locality, and robustness--make the centralized systems approach less appealing, however. For these reasons (and others), we believe that the

large integrated systems of the future will be built upon a distributed hardware base and that control will be decentralized. These systems will therefore use a paradigm of cooperating experts rather than one of masters and slaves. Our investigation of these types of systems is focused upon Distributed Sensor Networks (DSN). A DSN naturally contains a geographically distributed set of sensors (not all necessarily of the same kind), communications mechanisms, and processes to combine information, perform correlations, and draw inferences. A DSN may also include command and control components as well as providing for the fusion of DSN information with information gathered outside the system proper, i.e., intelligence. Thus, research into DSN provides a rich source of problems, the solutions to which enhance our ability to design and build the large integrated systems of the future.

Autopsy. The goal of the Autopsy project is to develop tools for translating programs from one language to another. Initial work is focusing on translation of CMS-2 programs into Ada. A related goal is the study of formal semantic definitions of programming languages and the development of usable tools for writing such definitions and for extracting properties from them. Eventually, it is hoped that translation systems might be built semi-automatically by using the formal semantic definitions of the old and new languages as inputs to a translator generator. The system under development is based on the premise that manual intervention will be necessary, but the automatic parts of the system should do as much as possible. A variety of tools will be available to the user to aid in the translation. At one end of the spectrum, an automatic translation tool will try to translate as much of the old code as possible. Constructs recognized as untranslatable will be left for the user to handle separately. In other cases, specifically ones related to timing or other non-apparent side effects, the code may be translatable syntactically, but may need separate correction later. In both cases, it is intended that an editor and possibly a pattern-directed transformation system will be available. Also planned is a limited verification system for comparing hand-written code in the new language with old code. A monitor for the system will maintain a history of the actions taken to translate an old program into the new language. The monitor will have facilities for stepping backward, i.e., "undoing," and for replaying selected steps.

Portable Packet Radio Terminal. ARPA is currently developing a Packet Radio Network (PRNet) that will provide a wideband data communication capability much like the

ARPANET but with the added dimensions of mobility and dynamic configurability. As this concept gains acceptance in the military services, fundamental choices will need to be made about mobile terminals for use with the system. The ISI Portable Packet Radio Terminal Project, begun in June 1978, addresses the terminal characteristics desirable at the user level, the kinds of interfaces and system architectures required to support them, and the impact of such issues on terminal designs and other portions of the system. The work is intended to result in a demonstration-level portable terminal to test and evaluate various solutions to the problems raised by extreme portability in the packet radio environment. We feel that a comprehensive scientific study of these issues is important to properly exploit, match, and couple terminal design to the communications attributes and special capabilities of the PRNet and to determine those user interface characteristics best suited to supporting the major user needs in a mobile tactical environment.

Multiapplication Support Terminal. The MAST project is a development effort to demonstrate the use of applications partitioned between terminal and host with reduced dependency on host and communications resources. The effort demonstrates how to take advantage of the capabilities of state-of-the-art terminals and examines the issues of application software preparation, user interface, host interface, and ARPANET interface. Special consideration is given to the user and network interface requirements posed by the National Software Works.

User-Dedicated Resource. This project provides encouragement and help to new ARPANET users who face an initial barrier because of their lack of previous on-line experience, the rapidity of system modification, 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 three TENEX and two TOPS-20 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 operates two TENEX and TOPS-20 systems at a computer center that is part of the command and control testbed at the Naval Ocean Systems Center, San Diego, California. As part of the Military Message Experiment, ISI also operates one TENEX system at Camp Smith, Oahu, on a 161 hour per week basis. The Institute provides 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, and economy of scale in operation.

1. PROGRAM VERIFICATION

Research Staff:
Ralph L. London
Raymond L. Bates
Roddy W. Erickson
Susan L. Gerhart
David R. Musser
David H. Thompson
David S. Wile

Research Assistant:
David G. Taylor

Support Staff:
Lisa Moses

PROBLEM BEING SOLVED

Our work concerns the process of constructing computer programs in ways that make it possible to verify that each program meets its specification. Given some description of a programming task, it is ultimately necessary to construct two things: a formal specification of the task and a program for carrying out the task. The central ingredient of our work is the ability to verify the program, i.e., demonstrate by a mathematical proof that the specifications and the program are consistent with each other. The style of specifications (axiomatic definitions plus derived properties of these definitions) and the methods of program construction are all important influences in making verification feasible. Our work emphasizes the construction of tools to aid verification, specifically (1) tools to translate programs into conjectures called *verification conditions*; (2) tools sufficiently powerful and natural to be guided by humans in proof attempts; and (3) tools to express and manipulate formal specifications.

One example from among numerous instances where it is vitally important that computer software perform according to its specifications is in the area of communication protocols. These are programs for establishing and maintaining communication between separated users over an unreliable medium with unknown delays. The proper operation of protocols is essential to such systems as electronic mail and electronic fund transfer. Lost or spurious messages cannot be tolerated. We are developing specifications of relevant properties of protocols, and we expect to verify several protocols against their specifications.

GOALS AND APPROACHES

In order to support such applications of program verification to critical software components, the goals of our work are to

- develop a comprehensive, well-integrated set of techniques for constructing fully specified and verified software components, from which large, reliable, and modifiable software can be economically produced;

- implement the techniques in a widely accessible, user-oriented, interactive computer system; and

- demonstrate the usefulness and practicality of the techniques and system in applications to real software where reliability is of critical concern.

Overview of the Affirm System

Our research at ISI has contributed significantly to the understanding of many fundamental issues underlying program specification and verification and to the development of effective techniques for dealing with these issues. These techniques include (1) a methodology for employing abstraction and hierarchical program organization and (2) extended system and language facilities, including improved specification and theorem-proving techniques. We have implemented many of these new techniques in an experimental verification system called Affirm.

The Affirm System accepts data abstractions and programs in a specification language based on the algebraic axiom method [3, 7, 9, 21] and in a programming language based on Pascal. Specifications of data abstractions can be tested for consistency and completeness (in a restricted sense) [20]. The Affirm verification condition generator supports the standard inductive assertion method. Affirm also contains a natural deduction theorem prover for interactive proof of verification conditions and properties of data abstractions. Affirm is the successor to, and combines features of, two previous systems developed at ISI, the Xivus System [8] and the Data Type Verification System [10, 18]. Further information and descriptions of Affirm may be found in an overview paper [6] and the *Affirm Reference Library* [23].

Specification Language

Affirm provides a specification language that has features in common with programming languages (such as its use of familiar conventions for expression syntax and declaration structure) and with logical languages (such as first-order predicate calculus). The use of data abstraction is supported by both the structure of the specification language and the capabilities of the theorem-proving component of the system. The system directly supports the algebraic axiom method of specification [11]. It also accommodates

abstract model specifications [26]. The user can define these abstractions (or access them from a library) as well as others that are particularly suited to his or her application domain. In conjunction with the theorem-prover commands, the user can build up a useful base of axioms and derived properties about data abstractions that can be invoked in later proofs. Those abstractions may become involved in proofs by their use in expressing either programs--e.g., a program that manipulates a queue or file--or specifications of programs. An example of the latter would be the use of inductive assertions to express relationships between variables used by a program, say, the use of sequences to describe the list of values processed during the execution of a loop. Appendix A presents an example of a data type specification in the form accepted by the system.

Programming Language

The programming language currently accepted by Affirm is a version of Pascal extended to include IMPORTS clauses as in Euclid [12], minor extensions to statement syntax and semantics, and a more uniform treatment of expression syntax. Statements may include embedded assertions expressed in the predicate language subset of the specification language. The verification specialist may introduce pre and post conditions on procedures and functions, invariant assertions and/or subgoal assertions on WHILE, FOR, and REPEAT statements, and ASSERTING clauses on GO TO and RETURN statements. The verification specialist may also introduce assertions at any point via the ASSERT statement. The parser for the language is a recursive descent parser generated automatically from a BNF-like grammar, a technique chosen to allow easy extension of the programming and assertion languages [25].

The verification condition generator is a straightforward implementation of a set of axiomatic rules expressing the definition of the language in terms of a predicate transformer similar to Dijkstra's weakest precondition transformer [4]. The substitution rules of the Euclid proof rules [15] are used, thus requiring the same restrictions on aliasing as in Euclid.

Theorem Prover

The theorem prover in the Affirm System is based in many ways on ideas and previous theorem-proving programs of Bledsoe and Bruell [1], Boyer and Moore [2], von Henke and Luckham [24], Lankford [13], and Suzuki [22]. The theorem prover operates on formulas

in the first-order predicate calculus that use functions defined in abstract data types. These formulas typically represent either verification conditions from programs being verified or desired properties of the data types themselves.

The theorem prover does not mandate a particular proof strategy. Commands are provided for the use of lemmas,[1] instantiation of variables, induction, and separate proof of parts of a theorem (subgoals). These commands result either in reduction of the current goal to TRUE or in one or more new subgoals whose truth implies that of the former goal. As each subgoal is generated it is simplified using rewrite rules [19] corresponding to axioms and previously derived properties of abstract data types.

The prover also provides a means by which an automatic completion of the proof process is attempted. This is the method of chaining and narrowing [14] based on a decision procedure for those valid formulas having a single ground instance that is a tautology.

As a proof proceeds, the system maintains a proof tree that records the steps taken. The status of subproofs or unproved assumptions is also maintained. When all the assumptions are finally proved, the proof is marked complete. The user may display this status information and print the goals and subgoals that have been generated.

Experience has shown that users can best visualize a proof when the complexity of expressions is carefully minimized. Indeed, this is the major task for substantial theorems. Provision is made, therefore, for the user to employ definitions in stating a theorem. These serve to "bundle up" potential sources of complexity until they are explicitly expanded. If desired, only some of the instances of a definition will be expanded. Additionally, it is possible to simplify and strengthen a goal by deleting subexpressions, subject to soundness constraints. (This technique can be useful in the proof of verification conditions, which typically have unneeded hypotheses.)

System Facilities

Affirm provides rudimentary facilities for inserting and retrieving data type specifications. Types *Boolean* and *Integer* are predefined in the system. (Currently, the

[1] Examples of the use of these and other commands appear in Appendix A.

system knows very little about integers.) An extensible library of common types is also provided. Currently, this library contains such types as sets, sequences, and queues.

At any point the user can freeze the current state of the system and restart it at a later time. The system also maintains a "history list" of user-specifiable length and contains general commands that redo or undo previous commands still on the history list.

IMPACT

We have successfully used Affirm to verify properties of a variety of simple data types and programs using them, e.g., sets, sequences, binary trees, and circular lists. Using existing capabilities of Affirm we have specified and verified a simple protocol without changing Affirm. On a larger scale, we have successfully applied the methodology using Affirm to specify and to verify properties of the file maintenance module of the Military Message Experiment [5]. System space limitations and inadequate support for managing proofs, both discovered during this activity, still preclude Affirm proofs of several key theorems, although informal proofs exist. Our decomposition of the problem included a high level specification of BLISS code, axioms of the data structures of the implementation, Pascal programs with abstract data types that simulate the actual BLISS code, and the abstract-to-concrete links between the Pascal and BLISS programs. On the basis of this experience we have demonstrated that our specification and verification methods and tools apply to real, intermediate-sized, moderately complex software components, and that we have the basis for a practical program verification system.

Verification concerns have had an impact on the Ada programming language. In addition to a number of ideas from programming languages such as Euclid and Alphard, whose goals include verified programs, the design and rationale of Ada include cases of perceived verification ease in making and justifying decisions. To assist in this, we participated in several Ada language reviews; in particular, we helped produce a verification-oriented evaluation of the Green and Red languages as part of the final selection process.

We have already seen the Affirm System used effectively by individuals (including some outside the project), thereby demonstrating the utility of verification as one means of understanding and managing the complexity of computer programs. For instance, we

specified and verified a "toy" security kernel in two days, with visitors from MITRE [16]. All our activity is directed toward a successful transfer of the ideas and capabilities of program verification to improve the quality and reliability of computer software systems. There is an increasing interest in program verification concerns in industry and government, as evidenced by the growing number of comparisons of verification systems [17].

FUTURE WORK

We will add to Affirm improved user aids, user interfaces, and methods of managing proofs such as inquiries about the interconnections of the parts (e.g., lemmas and definitions) of the proofs and the suppression or display of detail as is appropriate. We will augment the existing versions of the user's manual, reference manual, and annotated examples of Affirm use. Experiments in protocols, security kernels, and common verification problems are being designed and performed to determine the strengths and weaknesses of our methodology and technology.

We will build specifications and abstract programs of a communications protocol and derive behavioral assertions on the underlying protocol to function properly, and then verify the abstract programs. Briefly, the specification to be verified is that a copy of the communication exists at the destination or suitable exceptions are reported.

Finally, we will build a library of program components, each fully specified and verified, that can be combined hierarchically to form larger programs. Additional parts of the components include the necessary concepts, derived properties, and terminology for verification of the components. There are open questions on how to organize the components, how to state and prove a rather complete set of properties about a component, and how to provide the necessary assistance to users of our system. The notion of components will, of course, be used in the protocol work.

APPENDIX A: EXAMPLE OF THE USE OF AFFIRM

The following is a transcript of the use of AFFIRM on a common data type, set. Refer to Appendix B for the meaning of individual commands. Input is requested from the user by "U:" preceded by a number referring to an event (or command). Some events for extraneous steps have been deleted.

It is important to remember the cycle of theorem proving: the user commands some action, it happens or fails, and control returns to the user. If the command affects the current proposition, a normalization (rewriting and simplification) occurs before returning to the user.

```
Transcript file <GERHART>AFFIRMTRANSCRIPT.23-MAR-80;2
is open in the AFFIRM.system <AFFIRM>AFFIRM.SAV;100
```

First we will load in the main type of interest and any types it needs, here ElemType.

```
2 U: needs type setofelemtype;
 file created for AFFIRM on 20-Mar-80 19:05:43
SETOFELEMTYPECOMS
 compiled for AFFIRM on  7-Mar-80 12:01:52
 file created for AFFIRM on  4-Mar-80 00:23:26
ELEMTYPECOMS
(File created under Affirm 35)
<PVLIBRARY>ELEMTYPE.COM;1<PVLIBRARY>SETOFELEMTYPE.;3
```

These types have been processed earlier through the Knuth-Bendix algorithm, then stored as files in our library of data types, and are now loaded and ready for use. First let's print them and notice some points about each type.

```
4 U: print type ElemType;
type ElemType;
declare x: ElemType;
axiom  x=x == TRUE;
end {ElemType} ;
```

ElemType is just a dummy type with nothing known about it except reflexivity of equality.

```
7 U: print type setofelemtype;
(setofelemtype => SetOfElemType)
type SetOfElemType;

needs  type ElemType;

declare dummy, s, s1, s2, ss, t0: SetOfElemType;
declare ii, i, i1, i2, x: ElemType;

Interfaces
      NewSetOfElemType, s add x, s rem i, s diff s1,
      s int s1, s union s1: SetOfElemType;

Infix  union, diff, int, rem, add;

Interfaces
      i in s, isNewSetOfElemType(s), s subset s1,
      s=s1, Induction(s), NormalForm(s): Boolean;
```

```
infix  subset, in;

axioms dummy=dummy == TRUE,
       NewSetOfElemType = s add i == FALSE,
       s add i = NewSetOfElemType == FALSE,

       NewSetOfElemType rem i == NewSetOfElemType,
       (s add x) rem i == if x=i
                          then s rem i
                          else (s rem i) add x,

       NewSetOfElemType diff s == NewSetOfElemType,
       (s add x) diff s1 == if x in s1
                            then s diff s1
                            else (s diff s1) add x,

       NewSetOfElemType int s1 == NewSetOfElemType,
       (s add x) int s1 == if x in s1
                           then (s int s1) add x
                           else s int s1,

       NewSetOfElemType union s1 == s1,
       (s add x) union s1 == (s union s1) add x,

       x in NewSetOfElemType == FALSE,
       i in (s add x) == ((i=x) or i in s),

       isNewSetOfElemType(s) == (s = NewSetOfElemType);

define  s=s1 == (s subset s1 and s1 subset s),

        s subset s1 == all x (x in s imp x in s1);

schemas Induction(s) == cases(Prop(NewSetOfElemType),
                              all ss, ii (IH(ss) imp Prop(ss add ii))),

        NormalForm(s) == cases(Prop(NewSetOfElemType),
                               all ss, ii (Prop(ss add ii)));

end {SetOfElemType};
```

First notice right below the print command the evidence of spelling correction, putting in the capital letters in the name SetOfElemType. Very handy feature! Let us review our basic concepts of data types:

1. *The set constructors are NewSetOfElemType, representing the empty set, and Add, representing a set with an element added to it. Axioms for the operations are given in terms of each of these constructors, hence are defined for all possible sets.*

2. *The usual set operations are defined: union, intersection (int), difference (diff), element removal (rem), and membership (in). Axioms are used to define each of these operations.*

3. *Other usual set operations are subset and equality, which we have specified as definitions, not axioms, because of the free variable in the subset operation.*

4. *There are other ways to specify sets, e.g., making subset specified by axioms, but having an auxiliary defined version.*

First we will prove a small property of sets to illustrate the kinds of properties we can prove and that we often need in larger proofs. We first establish an environment for the proof.

```
10 U: edit Setofelemtype;
(Setofelemtype => SetOfElemType)
type SetOfElemType
dummy, s, s1, s2, ss, t0: SetOfElemType
ii, i, i1, i2, x: ElemType
```

The little property will express when an element i is in a set s with an element x removed from it, namely when i is already in s and is not x.

```
12 U: try RemEqv, i in s and i ne x eqv i in (s rem x);
Making node RemEqv a theorem.
RemEqv untried
all i, s, x (
  if i in (s rem x)
     then i in s and not (i=x)
     else i in s imp i=x)
```

eqv is printed in its if-then-else form:

```
a eqv b -> if b then a else not a
```

Our only real (or needed) tool for proving such a property is induction on the structure of s, as provided by the Induction schema of the type SetOfElemType.

```
15 U: employ Induction(s);
Case NewSetOfElemType: Prop(NewSetOfElemType) proven.
Case add: all ss, ii (IH(ss) imp Prop(ss add ii)) remains to be shown.
 (add:)
all ss, ii, i, x (
      IH(ss)
  imp if    i
         in if ii=x
              then ss rem x
              else (ss rem x) add ii
```

```
        then ((i=ii) or i in ss) and not (i=x)
        else (i=ii) or i in ss imp i=x)
```

(The 'cases' command is applicable)

16 U: **cases;**

```
all ss, ii, i, x (
      IH(ss)
  imp if ii=x
        then if i in (ss rem x)
               then      (i=ii) or i in ss
                    and not (i=x)
               else (i=ii) or i in ss imp i=x
        else if i=ii
               then not (i=x)
               else if i in (ss rem x)
                      then i in ss and not (i=x)
                      else i in ss imp i=x)
```

17 U: **replace;**

```
all ss, ii, i, x (
      IH(ss)
  imp if ii = x
        then if i in (ss rem x)
               then      not (i = x)
                    and i in ss
               else     i = x
                    or not (i in ss)
        else     i = ii
             or if i in (ss rem x)
                  then      i in ss
                       and not (i = x)
                  else      i in ss
                       imp i = x)
```

Notice that the replacement of equalities gave us several different logically equivalent versions of the induction hypothesis which is:

19 U: **print IH;**

```
      IH(s) == all i, x (
               if i in (s rem x)
                 then i in s and not (i=x)
                 else i in s imp i=x)
```

20 U: **invoke IH;**

```
all ss, ii, i, x (
  some i'(ss, i, x, ii),
       x'(ss, i, x, ii) (
  if i' in (ss rem x')
     then      i' in ss
          imp     i'=x'
               or if ii = x
                    then if i in (ss rem x)
                           then      not (i = x)
                                and i in ss
                           else     i = x
                                or not (i in ss)
                    else     i = ii
```

```
                    or if i in (ss rem x)
                        then     i in ss
                             and not (i = x)
                        else     i in ss
                             imp i = x
    else     i' in ss imp i'=x'
        imp if ii = x
            then if i in (ss rem x)
                    then     not (i = x)
                         and i in ss
                    else     i = x
                         or not (i in ss)
            else     i = ii
                 or if i in (ss rem x)
                        then     i in ss
                             and not (i = x)
                        else     i in ss
                             imp i = x))
```

This is a horrible looking expression, but we know that the induction hypothesis has the free variables i and x in it and that we want to identify these with the renamed variables that popped out of IH.

```
22 U: search;
1/4:  (x' = x) and (i' = i)
Proved by chaining and narrowing
using the substitution
(x' = x) and (i' = i)

TRUE
RemEqv proved
```

So search found the instantiation and the proof is done. Its internal record looks like:

```
24 U: print proof;
RemEqv is:      i in s and not (i=x)
          eqv i in (s rem x)
proof tree:
15:!   RemEqv
           employ  Induction(s)
       NewSetOfElemType:
         Immediate
16:     add:
         2    cases
17:       3    replace
20:       4    invoke  first  IH
22:       5    put  (x' = x) and (i' = i)  {search}
22:->     (proven!)
```

The numbers in the left column are event numbers. The other numbers are assigned to intermediate propositions for reference purposes. The arrow (->) shows the current point in the proof. Translating the proof into words: RemEqv was proved by Induction on s (according to a user-defined Induction schema). The basis case NewSetOfElemType was immediate (just from the axioms and simplification rules). The add case was proven by re-arranging the if-then-else structure of the proposition, replacing some equalities and using the induction hypothesis with its free variables renamed to those in the current proposition (automatically found by search). We could have generated this proof with several different orders of these same commands.

APPENDIX B: BRIEF COMMAND SYNOPSES

This list includes only those commands actually referenced in the transcript.

axiom *rule*
: Makes a rewrite rule L → R out of the rule L == R, and adds it to the system's rule set. All rules are applied to expressions during simplification, after each theorem-prover command.

cases Applies the special rule

f(if b then x else y) → if b then f(x) else f(y)

thus "raising" embedded ifs.

declare *ids*: *typeName*
: Declares the names to be variables of the indicated type.

define *rule*
: Makes a rewrite rule out of the equation. Definitions are not automatically rewritten during simplification; you must explicitly request application using the invoke command.

edit *typeName*
: Opens a new type context for subsequent specification commands. See the type and end commands.

employ *schemaName*
: Uses the schema to perform induction or case analysis.

end Ends the current type context and restores the previous one. Specification commands affect only the current type.

interface *op* (*params*): *typeName*
: Defines the domain and range information for an operation. *Params* are variable names, not type names.

needs type *typeNames*
: Causes the system to find and read the type specification for each specified type name.

print *option furtherArguments*
: Prints something. Common *options* are:

type *typeName*
: Lists the specification of the type.

IH Lists the current definition of the induction hypothesis IH.

proof *theorems*
: Lists the proof trees of the indicated theorems.

replace *expression*

Performs equality substitutions.

search

Attempts to find a set of instantiations of existential quantifiers that results in reducing the proposition currently being proven to true.

schema *rule*

Defines induction and case analysis schemas. See the description in the command synopses.

try *name, expression*

Attempts the proof of the named expression.

type *typeName*

Establishes a new context for subsequent specification commands.

REFERENCES

1. Bledsoe, W. W., and P. Bruell, "A man-machine theorem-proving system," *Artificial Intelligence* 5 (1), 1974, 51-72.

2. Boyer, R. S., and J. S. Moore, *A Theorem-Prover for Recursive Functions: A User's Manual*, SRI International, CSL-91, June 1979.

3. Burstall, R. M., and J. A. Goguen, "Putting theories together to make specifications," in *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pp. 1045-1058, IEEE, August 1977.

4. Dijkstra, E. W., *A Discipline of Programming*, Prentice-Hall, 1976.

5. Gerhart, S. L., and D. S. Wile, "THE DELTA EXPERIMENT: Specification and Verification of a Multiple-User File Updating Module," in *Proceedings of the Specifications of Reliable Software Conference*, pp. 198-211, IEEE Computer Society, April 1979.

6. Gerhart, S. L., et al., An overview of AFFIRM: A specification and verification system, 1980. (Will appear in *Proceedings of IFIP Congress 80*, October 1980.)

7. Goguen, J. A., and J. J. Tardo, "An introduction to OBJ: A language for writing and testing algebraic program specifications," in *Proceedings of the Specifications of Reliable Software Conference*, pp. 170-189, IEEE Computer Society, April 1979.

8. Good, D. I., R. L. London, and W. W. Bledsoe, "An interactive program verification system," *IEEE Transactions On Software Engineering* SE-1 (1), 1975, 59-67.

9. Guttag, J. V., E. Horowitz, and D. R. Musser, "Abstract data types and software validation," *Communications of the ACM* 21, December 1978, 1048-1064. (Also USC/Information Sciences Institute RR-76-48, August 1976.)

10. Guttag, J. V., E. Horowitz, and D. R. Musser, "The design of data type specifications," in Yeh, R. T. (ed.), *Current Trends in Programming Methodology*. Volume IV: *Data Structuring*, pp. 60-79, Prentice Hall, 1978. (An expanded version of a paper that appeared in *Proceedings of the Second International Conference on Software Engineering*, October 1976.)

11. Guttag, J. V., and J. J. Horning, "The algebraic specification of abstract data types," *Acta Informatica* 10, 1978, 27-52.

12. 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, 1-79.

13. Lankford, D. S., *Canonical Inference*, University of Texas Automatic Theorem Proving Project, ATP-32, 1975.

14. Lankford, D. S. , and D. R. Musser, On semi-deciding first-order validity and invalidity, 1978. (Submitted for publication.)

15. 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," *Acta Informatica* 10, (1), January 1978, 1-26.

16. Millen, J. K., *Operating System Security Verification*, MITRE Corporation, M79-223, September 1979.

17. Millen, J. K., M. H. Cheheyl, M. Gasser, and G. A. Huff, *Secure System Specification and Verification: Survey of Methodologies*, MITRE Corporation, MTR-3904, February 1980.

18. Musser, D. R., "A data type verification system based on rewrite rules," in *Proceedings of the Sixth Texas Conference on Computing Systems*, November 1977.

19. Musser, D. R., "Abstract data type specification in the AFFIRM System," in *Proceedings of the Specifications of Reliable Software Conference*, pp. 47-57, IEEE Computer Society, April 1979. (Also in *IEEE Transactions on Software Engineering*, January 1980.)

20. Musser, D. R., "On proving inductive properties of abstract data types," in *Proceedings of the Seventh ACM Symposium on Principles of Programming Languages*, , 1980.

21. Nakajima, R., *Sypes--partial types--for Program and Specification Structuring and a First Order System Iota Logic*, Institute of Informatics, University of Oslo, Norway, Research Report No. 22, November 1977.

22. Suzuki, N., "Verifying programs by algebraic and logical reduction," in *Proceedings of 1975 International Conference on Reliable Software*, pp. 473-481, ACM Special Interest Group on Programming Languages, April 1975.

23. Thompson, D. H., et al., *The AFFIRM Reference Library*, USC/Information Sciences Institute, 1979. (5 vols.: *Reference Manual*, *User's Guide*, *Type Library*, *Annotated Transcripts*, and *Collected Papers*.)

24. von Henke, F. W., and D. C. Luckham, "A methodology for verifying programs," in *Proceedings of 1975 International Conference on Reliable Software*, pp. 156-163, ACM Special Interest Group on Programming Languages, April 1975.

25. Wile, D. S., POPART: Producer of parsers and related tools, 1979. (USC/Information Sciences Institute report, in preparation.)

26. 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, 253-265.

2. MILITARY MESSAGE EXPERIMENT

Research Staff:
Robert H.Stotz
David Wilczynski
Steven Finkel
Robert Lingard
William Mark
Paul Raveling
Leroy Richardson
Jeff Rothenberg
Ron Tugender
Tom Wisniewski

Support Staff:
Andrea Putnam
Jeanne Ramirez

Contributing Staff:
Kyoo Jo
Vernon Dieter
George Dietrich
Nelson Lucas
Robert Parker

PROBLEM BEING SOLVED

The Military Message Experiment (MME) is a joint program funded by DARPA and the Navy to study the nature of automated interactive message processing in an operational military environment. The test, which ended October 1, 1979, was conducted at Camp Smith, Oahu, Hawaii, by a selected subset of users from Commander in Chief Pacific (CINCPAC) Operations Directorate (J3), which included the Command Center and J3 support groups. The focus of the experiment has been the SIGMA message system, developed at ISI, which runs on a PDP-10 under the TENEX operating system and connects to the AUTODIN network through the Local Digital Message Exchange computer (LDMX). The test environment includes 25 CRT terminals plus seven electrostatic printers, distributed in the Command Center and the offices of the J3 staff.

The purpose of the test is to evaluate the utility of an interactive message service and to study its impact on the user organization; in addition, the test provides insight into how future message systems for this type of environment should be designed and built. Experience in the ARPANET community has shown that on-line computer services can alter the basic fabric of communication within the user organization, depending on such variables as the community of users, accessibility of the terminals, difficulty of learning the system, reliability of the service, features offered, and response time. The goal of MME has been to learn the effect of these variables by studying the use of SIGMA in a command and control environment like CINCPAC. With this information, the various military organizations responsible for communications should be more effective in specifying the requirements for future production message systems.

ISI became involved in this experiment in 1973 when it performed a study of the communication needs on the island of Oahu [2]. In late 1973 the Information Automation (IA) Project was formed to investigate the application of automation to military communications. The concept of conducting an experiment in an operational military command evolved slowly. In December 1975 a Memorandum of Agreement was signed by DARPA, COMNAVTELCOM, COMNAVELEX, and CINCPAC, defining the MME. At that time three DARPA contractors, including ISI, were working on candidate message systems for MME. In February 1977 an extensive evaluation of the three systems was made, and, although still incomplete, ISI's SIGMA message system was selected. The PDP-10 computer and five user terminals were installed in May 1977 at CINCPAC, and SIGMA was brought up.

During the first 12 months of operation, the system was tested, shaken down, and gradually improved. A few "friendly" users were introduced to the system and their evaluation influenced the direction of subsequent SIGMA development. Experience was gained in how to train users, resulting in increased emphasis in on-line, self-contained training aids. By April 1978 SIGMA was receiving all of the J3 AUTODIN traffic (approximately 700 messages daily) on a 24 hour a day basis, 10 terminals were in users' offices, and use of the system by operational personnel was beginning. A phase of the experiment called Limited Experimental Use began in July 1978.

In October 1978 a KL processor was substituted for the older, slower KA processor and memory was expanded to one million words. At this time ISI took responsibility for operation and maintenance of the computer system.

In parallel with the development of the message service, the Navy and ARPA through MITRE Corporation defined the test objectives for MME and the plan for achieving them [3, 4]. This plan led to SIGMA's Data Collection Facility (DCF), which logs pertinent data about the use of SIGMA. This data is recorded in TENEX files on a continuous basis, and at regular intervals it is dumped from files to magnetic tape and sent to MITRE for analysis.

GOALS AND APPROACH

For the past 4 years the MME project has been developing the SIGMA message service to support the Military Message Experiment. The primary goal of this experiment

was to evaluate the utility of automated message handling in a military command headquarters. From the outset a number of subgoals were identified for the message service, including functionality, flexibility, response time, availability, reliability, ease of use, training support, and data collection support.

Functionality

The message handling in the Operations Directorate of CINCPAC is quite different from that of the ARPA research community. An AUTODIN message is considered a formal document "released" by the commanding officer of an organization (or one of his deputies in his name) and represents an official position. Such a message is always addressed to the commanding officer of another organization. Often the actual author is someone quite low on the staff who does not know his counterpart in the recipient organizations who will handle the message. Conversely, the recipient organization does not know who wrote the message. This formality leads to much review and coordination of draft messages at the originating command and a complex distribution problem at the recipient commands. Distribution involves assigning responsibility (Action) for the message and deciding who should get information copies. The usual approach is to distribute messages widely (requiring many copies) and to rely on informal internal communications to track action taken.

To satisfy the peculiar demands for coordination and distribution, it was considered necessary to design an entirely new message system based on a central message data base, making SIGMA different from most ARPANET message systems. A great deal of time was spent talking to users to learn their message-handling functional requirements. In addition, other groups involved in the MME (e.g., CTEC, MITRE) produced their own concepts of what was needed and submitted them to us for guidance and direction. Since this was to be an experiment, it was considered important to provide a lot of functions so the utility of each could be determined. As a result, a rich set of functions developed in SIGMA tailored to the military community.

It was understood from the beginning that these efforts to define the needed functions were only a starting point and that as the experiment proceeded, requirements for new, unforeseen features would arise. This requirement, to be able to adapt the system to the users' requests during the course of the experiment, led to another design goal: flexibility.

Flexibility

Because it was an experiment, the message service had to be designed to anticipate changes. This was one of the strong influences in selecting TENEX as the operating system. TENEX provided the software development environment necessary to allow continual evolution of the system. Flexibility dictated such features as using a high-level language for the development and making the command language processor table driven. We also tried to make the functionality of the system modular by defining message system primitives, called micros, that were combined to produce command level tasks. Configuration management procedures were instituted to control the release of new software at regular intervals. Throughout the development, rapid implementation was stressed rather than thorough design documentation.

Unfortunately this focus on flexibility was in direct conflict with another goal: user response time.

User Response Time

If the system is going to be used, it must respond in a reasonable time to user requests. Although the goal was clear, the actual design for good response time was not. As a result, the approach taken was to build the system first and measure and then improve response during its use. The performance issue first became evident during the evaluation and selection before the system was delivered to CINCPAC. Unfortunately, as changes were introduced to improve performance, they were often negated by functional enhancements being instituted simultaneously. Therefore, five months after the system was delivered it could only support four or five users, threatening the entire experiment.

Only through a massive effort that involved rewriting nearly every major component of the system, replacing the KA processor with a KL processor, and increasing memory from 256K to 1024K words was the problem contained. Even so, performance continued to be an issue since it limited our ability to add many more terminals.

Reliability

This goal, like response time, was recognized early, but its importance was not fully appreciated by anyone associated with the program. Before MME was defined and CINCPAC chosen as the host site, a system architecture was proposed to provide reliability and availability through redundant hardware and a fully backed-up, distributed

database. However, when the MME was defined, its experimental nature and a limited budget led to the decision that a single system with 25 terminals would have to suffice.

The equipment configuration chosen had redundant processors, pagers, and data channels, but not redundant memory or disk, the two least reliable components. TENEX was chosen for its flexibility in spite of known weaknesses in the file system. The normal problems of operating a computer were compounded in the CINCPAC environment by continual problems with power, air conditioning, and dirt. Finally, we did not realize how difficult it would be to maintain a continually changing, complex software system from a distance of 2500 miles while having no network access to the computer. These factors contributed to our inability to provide quite the 24 hours a day, 365 days a year service level.

This failure made some users unwilling to rely on the system and therefore affected the experiment's results.

Ease of Use

The typical user at CINCPAC has no computer experience. He functions in a world of paper: he keeps files of important messages, drafts new messages with a pencil at his desk, and has his secretary type them onto OCR forms for submission to AUTODIN. His message handling is an intuitive but small part of his activities. To use an automated system typically requires learning new procedures and techniques, a luxury we could not afford in the busy CINCPAC environment. SIGMA therefore had to be made as easy and natural to use as possible.

The SIGMA user interface was designed for easy use by noncomputer professionals. To achieve a natural interface the project developed a new interactive terminal, based on the standard HP 2649 hardware; firmware was developed at ISI (see [10]). The terminal supports two dimensional "what you see is what you get" editing. No conscious effort by the user is required to move data from the terminal to the host computer. A message is presented on the terminal as a window. The user merely scrolls the window up or down to see more data, regardless of how big the message is. While working on one message, a user may ask to view a second one in its own window. Other objects (files, text) may be quickly switched into view with a single button push. Function keys are provided to minimize typing for often-used commands.

No matter how natural this interface is, an automated system is different from a manual system and the users need training.

Training

As indicated above, CINCPAC users do not have time to take an extensive course in use of a new message system. It was always assumed that the principal training would occur on the job. To provide personal one-on-one training was impractical, so, in addition to providing the usual hardcopy reference manuals, SIGMA was designed to support on-line, self-guided training. These facilities come in three forms, Prompt (display of current command parsing information), Help (access to reference manual type of information), and Tutor (assistance in the form of on-line lessons and exercises). SIGMA's training facilities are described in [7] and [9].

Data Collection

Essential to conducting an experiment is an ability to collect results about the system's use. SIGMA produces a file for each user in which detailed information is recorded about every operation the user performs. These files are transferred to magnetic tape at regular intervals and sent to MITRE for analysis. In addition, SIGMA produces a separate Session Transcript of commands executed for the on-site staff to monitor abnormal terminations of user jobs, debug problems (showed the sequence of commands that led to an error), and monitor the use of the system.

PROGRESS

Status at the Beginning of the Reporting Period

At the start of the reporting period the MME system had been operating at CINCPAC for 16 months. During the first year of that period a long list of functional enhancements was implemented while at the same time the performance (response time) of SIGMA was improved by a factor of three. Then, finally, the background processes (Daemons), a source of many problems, were completely rewritten to make them stable and maintainable. During this time a few "friendly" users provided feedback on how SIGMA could be improved functionally.

In the spring of 1978 classified messages began flowing steadily to SIGMA, causing the database of real messages to grow. J301, the administrative office for J3, began using SIGMA to assign Action and Info on messages. The full complement of terminals

was delivered and approximately 100 users were introduced to the system. Though functional improvements were still to be made, Limited Experimental Use began in the summer, at which time the system was able to support about ten users with reasonable response. Still troublesome was the unreliable operation of the 10-year old computer system.

To improve the performance and reliability, a complete replacement of the KA processor with a KL processor with 1 million words of memory was scheduled to take its place in October. Along with the change in equipment, the operation and maintenance contract was moved to ISI. A full staff of civilian operators, hardware maintenance crew, and on-site system programmers was hired. For the first time MME had an adequate staff on-site to operate the machine.

Progress during the Reporting Period

As with any large computer system, it took several months to shake down the new hardware with its attendant new programs and operator procedures. There was an immediate improvement in system response--a benchmark load test was run with 20 users and the performance was deemed adequate.

In early January 1979 SIGMA Release 2.2 was delivered. It included message Readdressal, a revised memo format, Alerts, an improved form of Coordination, better error handling, and a number of bug fixes. After this release, CINCPAC allowed messages originating in SIGMA to be sent to the LDMX for transmission over AUTODIN.

With these functions added to SIGMA, Full Experimental Usage began in February. The user community was encouraged to use SIGMA for as much of their message handling as practical--some users eagerly learned how to use the system, others were reluctant to try the system at all. Though improving, system reliability was still a source of complaint from the users. One particularly annoying problem was running out of disk space, a situation that neither TENEX nor SIGMA handled gracefully. We finally began to appreciate the need for round-the-clock service and virtually 100 percent up time. This level of service has never been expected from an experimental system. However, the users who put all their files on-line and committed to on-line procedures for getting their job done were understandably unforgiving.

The reliability issue focused our attention on making SIGMA more robust and maintainable. We could not make TENEX and SIGMA 100 percent reliable, but we could improve our recovery procedures. A series of mini-releases (2.2, 2.22, 2.23) and a revision to the terminal firmware were sent out during the period February through April 1979 to improve this situation. Enhancements included a more robust communications protocol with the terminal, a terminal dump program, a program for rebuilding user state files, some directory verification programs, improved error reporting and recovery facilities, and a capability to run SIGMA under the TENEX Exec.

In March the CINCPAC SIGMA system was used during a two week, world-wide communications exercise, though it was not trusted to be the primary message handling facility (normal manual processing was used). Instead an additional action officer was assigned to the Crisis Action Team to "manage" exercise traffic on SIGMA to make pertinent information available on-line. Members of the Crisis Action Team were introduced to SIGMA (many came from directories other than J3 and had never seen the system) and were invited to use it as they saw fit.

Unfortunately, the system was down for several extended periods during the exercise, so its utility was not well tested. Some useful observations were made, however. The ease with which totally new people were able to pick up the basic system was very encouraging. One user from J4 was so enthusiastic about using SIGMA for generating outgoing traffic that he was assigned to be a regular MME user; he used the system extensively thereafter. During the exercise, the increased traffic load and message handling activity slowed SIGMA's response to a point where its use by other users had to be curtailed. It was found that messages that accumulated while SIGMA was down would deluge the Crisis Action Team when the system came back up. This exercise pointed out the need for specialized procedures for automated message handling during a crisis.

During April CINCPAC was asked whether it wished to retain the system after the end of the experiment. The response was a very lukewarm endorsement, pointing to problems with reliability of the system and limitations in usefulness because it only supported 25 terminals. Curiously, about this time a bad main power filter was discovered and its removal eliminated a major source of memory failures. At this same time a double density disk system was installed. This extra storage (now a total of

nearly 400,000 pages) allowed keeping 30 days of traffic on-line (up from 12-14 days) and eliminated the problem of running out of disk space. The reliability improvements these changes introduced were recognized too late. With CINCPAC's senior management's weak endorsement, ARPA made the decision in May to remove the system at the scheduled conclusion of the experiment, October 1, 1979.

The last functional changes were incorporated in Release 2.3, which was delivered in late May 1979. These changes, primarily suggested by users, included a facility to highlight text, an extension of the Comment feature for files, a capability to sort a file by Date Time Group, an EMPTY command to clear the contents of a specified file, and a special access control mechanism applied to incoming AUTODIN messages. This release was installed smoothly.

In June, as the last functional release was being tested, our first effort at technology transfer took place. An entire session of the National Computer Conference (NCC 79) was dedicated to MME. Five papers by MME project members were submitted and accepted for publication in the proceedings [9, 10, 11, 13, 14]. The session itself comprised two presentations covering the five papers, a movie on the MME, and a five member panel discussion.

Despite the decision to remove the system from CINCPAC in October, a last push was made to improve system reliability. ISI staff members were rotated on-site for one- or two-week periods through July and August. During this period, four more SIGMA mini-releases and two firmware changes were delivered to fix bugs tracked down by the on-site personnel. Reliability of the system was also improved by installation of better multiplexors and a fix to some PDP-11 communication hardware.

As MME drew to a close, several special experiments were scheduled. In August, special emphasis was put on using MME for message preparation, coordination, and release. The goal was to have 50 percent of J3's outgoing traffic originate from SIGMA; however, for the designated two week period only 59 out of 193 messages were released through SIGMA.

In the second week of September a special three-day CINCPAC communication exercise was scheduled. This exercise was a rerun of the exercise conducted in March

using the same messages that had been received at that time. These messages were retrieved from SIGMA's archive and processed to remove the action and info assignments that had been made on them previously. During the new exercise these messages were treated as new incoming messages and were forwarded to the Crisis Action Team. For this exercise new procedures were adopted for the Crisis Action Team using SIGMA exclusively for all their message handling. Seven terminals were provided in the simulated command center, one for each member of the team. In addition the Deputy Crisis Action Coordinator monitored the progress of the exercise from another terminal in his office. The exercise messages were injected through an exercise controller at an initial rate of 12 per hour and then stepped up gradually to a sustained rate of 35 per hour, which was three times the rate of the traffic during the original exercise. Although analysis of the test data is not complete, the special exercise clearly demonstrated that SIGMA was effective in a crisis situation with that level of traffic.

As the reporting period ended, the experiment officially concluded. Interestingly, there was a last minute effort by CINCPAC to reverse its earlier position and try to retain MME. A request for money to support the system was sent to JCS, citing improved reliability and better understanding of the system utility. This endorsement was made too late to justify changing plans.

With the conclusion of the on-site experiment approaching, the project team terminated further software development and began the final phase of the experiment--writing a final report.

IMPACT

As the experiment draws to a close, many people are addressing the questions of what has been learned and how can these lessons be applied to future automated message systems. It is generally agreed that MME clearly demonstrated the utility of automated message handling in an operational military environment. It also showed, however, that this utility is dramatically affected by some critical parameters, such as system reliability, response time, amount of coverage of the organization, functional performance, and amount of data storage. In addition, much experience was gained in conducting an "experiment" in an operational environment. Finally some issues have been raised by our experience that have a bearing on all DoD automation attempts. A

set of final results are currently being compiled, but the following represent some of the significant issues uncovered by the MME so far.

DoD Policy

There is an increasing awareness within DoD that complex computer applications such as message handling, command and control, and management information are currently not well enough understood to allow building effective systems. The introduction of automated tools into these complex areas can produce unpredictable results, both good and bad. Systems of this sort should be introduced in an "evolutionary" manner: as their impact is evaluated, they can be modified to better accomplish their mission. This is not to say any particular installation must be continually changing, but a mechanism to gradually evolve and improve the basic system must be provided. Experiments like MME are an excellent first step in such an evolutionary cycle. However, some efficient means for integrating the experiment results into the next generation of products still needs to be developed.

Incremented delivery of a communication system is difficult. We learned from MME that if a message system does not link enough of the right people, it is only marginally useful. It is clear that if terminals had been provided to all action officers at CINCPAC, MME would have been much more valuable to everyone. This would be extremely expensive and its cost would be hard to justify. If the system is installed gradually, however, the community must operate for an extended time with some users on-line and others not. Again MME showed this to be a difficult way to function, often requiring extra work (e.g., keeping both paper and electronic files of everything).

Another high-level policy issue that MME pointed up is the delivery of reliable computer cycles. Our experience at CINCPAC showed both the the necessity and the difficulty of operating a large complex computer system in a typical operational military environment. We encountered every imaginable problem, including power that fluctuated and even dropped without notice, crypto communication that could not stay synchronized, dirty air, air conditioning that was sporadic, fungus growth on components, restricted access to the computer floor because of security--and this was only in the computer center. The conditions in the user spaces were even less controlled. Security controls severely limited access to the user spaces to handle hardware or operational problems with the system. Since the trend in computer systems is toward more distributed

processing (e.g., "smarter" terminals, local printers, personal computers), maintaining such equipment and the systems that run within them in a typical military installation will continue to be a problem in the future.

How to Conduct an Experiment

A number of issues arose that speak to the general problem of trying to conduct an experiment like the MME. For example, we found it is crucial to motivate the users to the goals of the experiment. It was entirely reasonable for the users to simply view the system as a tool. When they found it flawed, many preferred to retreat to old methods rather than contend with the weaknesses of the new system. Because a critical mass of users was a necessity, this retreat affected those who were willing to contend with the difficulties. Sharing of files, coordination, intra-office communications are just some of the concepts needing total cooperation in order to be tested.

The classified nature of the operational site together with performance implications dictated that the MME software be developed on a separate computer. This separation meant that the developers were a long way from the experiment. Bugs were nearly impossible to trace because security denied us a remote connection. From the beginning we were forced to send our ISI programmers to Hawaii. This slowed our software development; not only was the missing man not able to program, but others often needed to interact with that person. We trained an on-site representative, hoping he could debug for us, but this proved to be wishful thinking--SIGMA was too large for a new person to assimilate so quickly. Although he was extremely effective in representing our interests, he simply could not know enough to classify each problem that came along. For the life of the experiment there was a continual need for SIGMA programming expertise on-site, a need we could supply only sporadically. A remote attachment to the system would have been a much more satisfactory solution; we then could have responded faster to problems by applying all of our available expertise rather than hoping that the person we sent knew enough to track down the difficulty at hand.

In experimental systems, changes are made to meet the evolving understanding of the application. This instability makes the systems inherently less reliable. Yet, as we have noted, reliability is essential in operational environments even for test systems. SIGMA was tested before each release by having five of our people exercise a cooperative two-hour scenario designed to test most of the functionality. As the system changed,

these scenarios were changed. This minimum test did not stress any of the system limits and provided only a cursory examination of many functions. We considered building semi-automated test facilities for SIGMA, but were discouraged by the magnitude of the job. Program verification research offered no help, and we do not see how a better programming environment would have served our purpose. In large evolutionary systems it is imperative to include in the design good test facilities from the beginning.

One final issue that has bearing on future experiments is defining the user's relationship to this test system. If the system is successful, some users will learn to depend on it. When the experiment is over, either the system must stay or the users must return to their old ways. Both have severe implications: keeping the system may be an expensive proposition (an experimental system is not likely to be cost effective). However, users are not motivated to learn a temporary system, a fact that may undermine the whole basis of the experiment. This problem needs to be considered at the outset of any future experiment like MME.

Automated Message Handling Systems

In ISI's Final MME Report, a portion of which will be included in the Navy's Final MME Report, our MME experience will be detailed. Though we are still sorting out the lessons and how best to express them, the following represent some important points that we plan to discuss:

Critical Mass of Users

At CINCPAC there were 22 terminals in user spaces, plus 3 more in commonly accessible areas (training and conference rooms). Response time during the early rush (around 7 a.m.) was marginal. So even though we could have provided more terminals, the 25 were a form of system limit. When the MME equipment configuration was defined we estimated that 25 terminals would provide a useful test; it became evident that this number was too small to effectively determine the utility of the system. An analogy with the telephone illuminates the issue: The more phones (or terminals), the more useful such a device becomes, because more people can be reached. In addition, with a message system, the more users contributing to the database, the more information it contains and the more valuable the service is. Further, if a message is to go to both on-line and off-line personnel, the message must be processed in different ways, often

requiring double work. A message service must support some "critical mass" of users to be worth using at all.

The comments of some users during interviews at the conclusion of the experiment suggest that MME was below a critical mass threshold for CINCPAC, though the ideal number remains a matter of conjecture. It was once estimated that 65 terminals would cover the J3 Directorate adequately, while some 200 terminals would be needed to cover the entire headquarters. Future Automated Message Handling Systems (AMHS) must be designed to handle these kinds of numbers, and cost justification should be considered on this basis.

Critical Mass of Messages

Just as important as the need for a critical mass of users is that for a critical mass of messages. If the user receives all of his message traffic in paper form but only part of it over the AMHS, he is hard pressed to see why he should use the automated system. MME was designed to include top secret traffic, but the CINCPAC communications center was not able to deliver these messages to SIGMA. Special intelligence traffic was never intended to be put on-line. But many of SIGMA's users worked with large numbers of these classified messages and found their absence from SIGMA very frustrating. Rather than deal with a system that allowed them to do only some of their work, they often chose to stick with manual processing. We also found that when disk size limited SIGMA to keeping 12 days worth of traffic, on-line users were not happy. After we began keeping 30 days of traffic on disk, complaints about archive retrieval vanished.

Critical Mass of Functions

SIGMA provided the CINCPAC users a rich set of functions that spanned message reading, word processing, and database management. SIGMA was purposely designed to offer a wide range of functionality in hopes we could ascertain which functions were most useful without biasing the user toward a particular style of use. It turns out that nearly all functions were used--different people used the system in different ways. J301 was almost the only user of the ROUTE instruction, yet his role is so critical that the experiment could not get under way until ROUTE was implemented. Some users relied heavily on the file search features; others, surprisingly, not at all. If any of the functional areas had been slighted, the set of users who took advantage of those

functions might have been discouraged from bothering with the system. The benefits of automating message handling seem to accrue from the integration of a wide range of functions into a single system.

Critical Mass of System Availability

An AMHS must be usable when the user wants it. MME suffered greatly because too often it was not available when a user needed it. We thought 95 percent uptime of a scheduled 161 hours per week would be sufficient for an experiment. This estimate was wrong. An AMHS must be available 24 hours a day, 365 days a year. Providing this sort of reliability is a major challenge.

What is Automated Message Handling?

Few people appreciate the complexity of message handling. A system that simply determines how to route incoming messages is not message handling. Neither is one that allows users to read their messages on-line rather than on paper. The payoff seems to be in combining the entire spectrum of message handling and word processing functions into a single system; it may be more appropriate to think of these as "Office Automation" systems than just as communication systems. The combination offers wider, faster, more accurate dissemination of information. Although messages form an appropriate base, it would be a significant mistake to ignore the other forms of communication offered through shared data.

FUTURE WORK

The Military Message Experiment project at ISI will continue for the first three months in FY80. The primary task is to write a final report describing the functionality and design of the SIGMA system and explaining the lessons we learned during the experiment. At the same time we will also transfer the technology we have accumulated through other means, such as writing papers, giving briefings, and assisting military organizations to document their requirements for automated message handling.

The current plan is for the final report to be organized into four sections, an introduction, a functional description, the design of SIGMA, and lessons learned. Since we expect to produce about 200 pages, we will bind each section of the report into separate volumes. We anticipate a portion of our final report will be integrated into the MME Program final report to be published by the Navy. Our report will be finished before

the Navy's, so the Navy can either use our report directly or extract the parts it wishes to use.

The normal mechanism for operating military commands to acquire new systems is to submit a ROC (Required Operational Capability). The Military Message Experiment evolved from a CINCPAC ROC for on-line message handling capability. CINCPAC is planning to update that ROC in light of its MME experience. U.S. European Command (CINCPAC's equivalent in Europe) is also preparing a ROC for text message handling. We anticipate helping these organizations to write a comprehensive document reflecting the lessons we have learned.

Finally we also expect to apply our experience in introducing interactive computing services to a military command headquarters. Several ARPA programs recently undertaken (e.g., Fort Bragg, Strategic Air Command) will be facing many of the issues that confronted the MME. We will provide guidance to these programs.

REFERENCES

1. Ames, S. R., and D. R. Oestreicher, "Design of a message processing system for a multilevel secure environment," in *Proceedings of the National Computer Conference*, AFIPS, 1978.

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

3. Goodwin, N. C., J. Mitchell, and S. W. Slesinger, *Test Plan for Military Message Handling Experiment*, Volumes I and II, MITRE Corporation, MTR-3268, July 1976.

4. Goodwin, N. C., and S. W. Slesinger, *Test Procedures for Military Message Handling Experiment*, MITRE Corporation, MTR-3521, October 1977.

5. Heafner, J. F., and L. H. Miller, *Design Considerations for a Computerized Message Service Based on Tri-Service Operations Personnel at CINCPAC Headquarters, Camp Smith, Oahu*, USC/Information Sciences Institute, WP-3, September 1976.

6. Oestreicher, D., P. Raveling, and R. Stotz, *HP/MME Terminal - Application Specification*, USC/Information Sciences Institute, TM-78-10, March 1978.

7. Rothenberg, J. G., *An Intelligent Tutor: On-line Documentation and Help for a Military Message Service*, USC/Information Sciences Institute, RR-74-26, May 1975.

8. Rothenberg, J., *SIGMA Message Service Reference Manual*, USC/Information Sciences Institute, WP-8, November 1977.

9. Rothenberg, J., "On-line tutorials and documentation for the SIGMA Message Service," in *Proceedings of the National Computer Conference*, AFIPS, June 1979.

10. Stotz, R., P. Raveling, and J. Rothenberg, "The terminal for the Military Message Experiment," in *Proceedings of the National Computer Conference*, AFIPS, June 1979.

11. Stotz, R., R. Tugender, D. Wilczynski, and D. Oestreicher, "SIGMA--An interactive message service for the Military Message Experiment," in *Proceedings of the National Computer Conference*, AFIPS, June 1979.

12. Tugender, R., and D. R. Oestreicher, *Basic Functional Capabilities for a Military Message Processing Service*, USC/Information Sciences Institute, RR-74-23, 1975.

13. Tugender, R., "Maintaining order and consistency in multi-access data," in *Proceedings of the National Computer Conference*, AFIPS, June 1979.

14. Wilczynski, D., R. Tugender, and D. Oestreicher, "The SIGMA experience--A study in the evolutionary design of a large software system," in *Proceedings of the National Computer Conference*, AFIPS, June 1979.

15. Wilczynski, D., R. Stotz, R. Tugender, and R. Lingard, "Message system architecture--Experience at CINCPAC with the SIGMA System," in *Compcon Spring '80*, IEEE, February 1980.

3. COMMAND AND CONTROL GRAPHICS

Research Staff:
Richard L. Bisbey II
Benjamin Britt
Dennis Hollingworth
Gertrud Mellstrom

Consultant:
Danny Cohen

THE PROBLEM BEING SOLVED

A key requirement for effective command and control (C2) is the accurate and timely exchange and effective presentation of relevant information at all levels of command. The application of advanced information processing techniques makes it easier to manage and augment this information exchange, permitting military personnel to better interpret and respond to ordinary as well as crisis situations. In particular, graphic communication of two-dimensional information, such as maps and charts overlayed with relevant tactical and strategic data, can greatly enhance both the quality and the efficiency of communications and provide added depth to the information exchange.

Computer graphics is not new to the military. Immediate or anticipated military requirements have quite often been the basis for technology development and deployment; this has been true in various areas of computer graphics. The military has employed computer graphics in a variety of special-purpose applications. However, the utility of graphics as a communications vehicle suggests that graphics be made available in a more generalized way, much as voice communication is made available via the AUTOVON system. This has become both attractive and economically feasible with the advent of digital computer communications networks such as the ARPANET and its planned military counterpart, AUTODIN-II, now under construction.

To explore this need, ARPA has contracted with ISI to design and develop a graphics system that could exploit the distributed processing capability intrinsic to a computer network, adapt to the changing communication and processing resources available during and between crisis situations, and evolve to meet future command and control processing and display requirements. In addition, ISI is developing representative application software that uses the graphics system to demonstrate the utility of a distributable, display-device independent graphics capability for command and control applications.

GOALS AND APPROACH

Crises are generally unpredictable and may require quick response to a rapidly changing situation. Because the nature and amount of the resources available to meet a crisis may vary, a C2 graphics system must be adaptable to the resources at hand, including processing, communications bandwidth, and display equipment. Furthermore, it should be extensible so that it can evolve in a fashion not traumatic to its users to accommodate new display devices or new communications technologies. These requirements resulted in five specific goals for the C2 graphics system.

1. The development and use of graphic applications programs should be independent of the display device type, with the graphics system adapting to and supporting terminals of varying capability. The developer of an application should be able to produce the application unmindful of the specific display device that will ultimately be used. He should also be able to test and debug the application program on whatever display device is at hand or is otherwise accessible and reasonably convenient to the test environment. Finally, the application program should be able to exploit the features available with the particular device to which it is connected without being limited to the greatest common subset of features of the terminals used; the graphics system should perform the requisite mapping of the requested capability to the features available.

2. The graphics system should be adaptable to the processing and communications resources available when the application/display device connection is established. The unpredictability of the communications bandwidth available and the location of and accessibility to computational resources during a crisis situation, together with the need to optimize use of available resources to meet the situation, suggest that the graphics system be tailorable to a wide variety of processor/communication combinations.

3. The graphics application should be oblivious to the placement of graphics system components and the requisite communications established to support their placement. Whether the graphics system is distributed across several computers to take advantage of processing or communications resources or whether the entire system resides within a single processor should be invisible to the application program. Any special machinations necessary to establish and support a particular configuration should be handled by and within the graphics system itself with any side-effects constrained to the graphics system, with the possible exception of system performance.

4. The graphics system should support the incorporation of possibly independently generated graphics pictures into an application program as

well as the creation of picture descriptions for use outside the immediate application environment. The graphic equivalent of a file mechanism should be provided for sharing of graphic information between arbitrary graphic application programs. Furthermore, since the display device available to the application that creates a graphic file will likely be different from those available to the programs using the resulting pictures, the files must be created and stored in a display-device-independent format.

5. The graphics functions available to an application should be sufficiently general to support a variety of application domains. This generality should include system configurability, communications adaptability, and device-independent multi-terminal usage. The user interface itself should provide a sufficiently rich set of graphics primitives such that highly tailored, application-specific graphics environments can be constructed on top of the interface without requiring alteration of the functionality of the graphics system itself.

The approach adopted to meet these goals was to develop a distributable, display device-independent vector graphics system in which graphic application programs could be written without regard to the particular display device upon which the output would ultimately be displayed. The system achieves display device independence by providing the application program with a set of generic, device-independent, two-dimensional vector graphic primitives (Graphics Language [1]) by which pictures can be described and interacted with at the application level. The application program deals with graphics in terms of named picture elements, called segments. Segments can be created, destroyed, merged, made visible or invisible, made touch-sensitive, and highlighted. Segments themselves are specified in terms of generic graphic primitives, including vectors, arcs, dots, text, and filled sectors and polygons, that may vary according to color, intensity and type-face.

Graphics Language (GL) defines the set of application independent functions for performing the above operations. Since several programming environments are expected to be used in the Command and Control environment (mainly FORTRAN, but also LISP, BLISS, MACRO and other languages) GL is defined (and implemented) in such a way that it can be easily used by application programs written in any of them.

The binding of the application program to a particular display device is deferred until program execution time, when the generic graphics primitives are mapped by the system

into specific operations and display modes appropriate to the device selected. The quality of the resulting picture is limited only by the capability of the display device.

The graphics system is distributable across multiple host computers interconnected by a communications network (such as the ARPANET). This allows the graphics display device to be located away from the graphics application program and allows the computational load introduced by the graphics system to be distributed across multiple processors, balanced to the computational resources and communications bandwidth available. The graphics system achieves distributability by virtue of its modular design and implementation, i.e., the system consists of a series of isolatable functions that communicate via a common communications mechanism (see [2] for more details).

The design separates the issue of system functionality from communications. In other words, the functionality of the system does not depend on the physical processor on which a given function resides during a graphics session. The physical location of graphic functions (as well as the display device) affect performance issues only; they have no effect whatsoever on the application program. Moreover, any intraprocess/ interprocessor communications mechanism might be used for communications between functions of the graphics system (including telephone, radio, or digital network/ internetwork links).

Modular organization of the graphics system has several benefits in addition to distributability. It allows the system to be easily extended to support other application languages and display device types. The former is accomplished by replacing an application interface component by one which interfaces to the new language, the latter by replacing the component that generates device orders. Such replacement has no effect whatsoever on the application programs or the fashion in which application programs use the graphics system and graphics language.

Modularity also allows new graphic functions to be easily added to the capabilities of the basic system. The architecture supports a single-terminal-to-single-application graphics environment. Additional graphic functions may be added to support more complicated configurations, including multiple programs concurrently connected to a single terminal and a single application program simultaneously generating graphics output for and accepting input from multiple (possibly dissimilar) display devices.

PROGRESS

Graphics System

In July 1979, ISI delivered the distributable (Level 2) version of the graphics system for TENEX/TOPS-20. The initial release of the Level 2 system supported the Genisco GCT-3000 raster-scan, bit-map color graphics display as well as the Tektronix 4010 series monochromatic display. Subsequently the HP-2648 monochromatic and Tektronix 4027 color graphics terminals were also supported. As described in the system architecture document [2], the system consists of seven major components: the Language Interface, Application Interface, Model, Clipper, Normalizer, Segment Pool Manager and Device Order Generator. The first five modules compose the "frontend" and are linked with the user application program. The remaining two modules constitute the "backend". Frontends and backends may reside on different machines; they communicate with each other using Graphics Protocol over the ARPANET.

In September, two additional components, a File Input Manager and a File Output Manager, were added to the system to support application program software under development. These components support the creation and playback of display-independent graphic files both during the current session and later.

As an extension of the basic mapping package, work was initiated and completed on a 64K vector map that included political as well as geographic boundaries. Information was encoded in such a fashion as to allow color coding of individual countries and power blocs. To satisfy an immediate need, a gateway program was written to allow the graphics system and a Tektronix display to be used with the Packet Radio Net at SRI.

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.

Concurrent with the development of the graphics system, ISI is developing several command and control applications to demonstrate various capabilities of the graphics system. During the current fiscal year, the bulk of our efforts in the application software area involved the design and prototyping of a Briefing Aid system which supports the creation of a briefing composed of graphic files (described above) and subsequent delivery of the briefing to several participants.

The package facilitates composition, review, revision and presentation of a briefing, using computer graphics display hardware. Such a computer graphics briefing offers 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.

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 graphics 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.

FUTURE WORK

Future research and development will focus on the following four 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 briefing aid capability.

4. Maintenance and improvement of graphics language and graphics protocol.

Situation Display

The present Situation Display will be expanded to allow displayed pictures to be saved on disk for later review. Situation Display will be augmented to utilize new facilities made available by LADDER, such as alerting. All documentation of Situation Display will be completed, and the final version delivered. Figure 1, on the following sheet, shows examples of Situation Display output.

C2 Map Capability

The present map capability is an interim facility intended to address immediate requirements associated with particular C2 applications. To improve this interim capability, ISI will deliver a higher resolution (65K vector) map and a multi-level user callable map display package which switches between map resolutions based on the area being displayed. ISI will also continue its interaction with DMA to determine their present capability and future plans and time-scale with respect to the type of data they will make available (detail, aggregation levels, projection types) and how maps will be made available (tape, on-line network access, video disk, etc.).

Briefing Aid

As indicated in an earlier section of this report, ISI is developing a Briefing Aid application program that will allow users to compose, review, and present briefings using computer graphic display hardware instead of conventional film media. Future work in this area will focus on development of a dynamic briefing aid system (including both preparer and viewer programs), which interfaces to Situation Display.

Graphics Language/Graphics Protocol Development

As an extension of our current efforts on the Level 2.0 system, one additional device driver will be delivered with our activities focusing primarily on maintaining and improving the system in response to comments from the user community and assisting The Rand Corporation in implementing Graphic System backends on the PDP-11. Considerable effort will also be directed toward extending the Level 2.0 system to support additional features associated with the Level 3.0 system in the area of multiple, non-homogeneous-display-device support, display-device multiplexing, etc. [2].

REFERENCES

1. Bisbey II, R., D. Hollingworth, B. Britt, and G. Mellstrom, Graphics language (Version 2.0), USC/Information Sciences Institute, June 1979.

2. Bisbey II, R., and D. Hollingworth, A distributable, device-independent vector graphics system for the military command and control environment, USC/Information Sciences Institute, April 1978.

3. Newman, W. M., and R. F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill, New York, 1973.

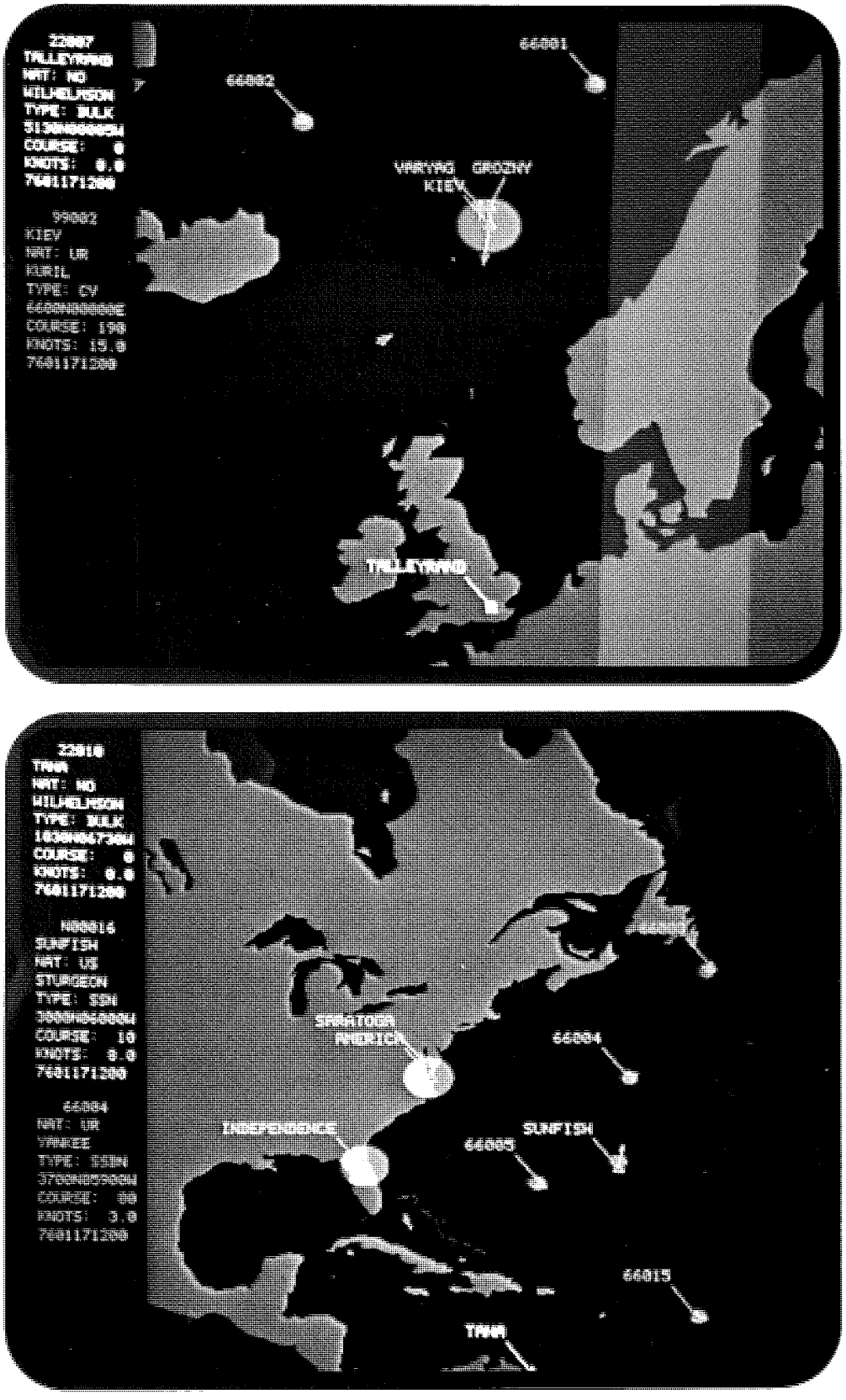


Figure 1. Typical situation display outputs using Graphics System.

REPO

sion

4. SPECIFICATION ACQUISITION FROM EXPERTS

Research Staff:
Robert Balzer
Neil Goldman
Lee Erman
Phil London

Research Assistants:
Chuck Williams
Mark James

Support Staff:
Joan C. Nosanov

PROBLEM BEING SOLVED

For several years the SAFE project has been exploring a unique approach to simplifying software specification. Rather than developing a "better" specification language, we have developed a prototype interface between a software specifier and the formal specification to be created. This interface embodies our knowledge of what constitutes good formal software specifications, and it attempts to restate the user's *informal* software specification in the required formalism.

We believe such an interface is necessary because formal mechanisms are foreign to man's nature. Only with considerable pain, difficulty, and expense can people be taught to use such formations, but they always revert to informal structures wherever possible. Even mathematical proofs are only partially formalized arguments that a line of reasoning is correct.

Formal software specifications must be produced, but that doesn't imply that people should create them. Rather, we believe, people should continue to write informal specifications (such as existing B-5 military software specifications) and use a computer-based tool to produce the formal software specifications. Such a division of effort between man and machine seems much more appropriate, with the computer system responsible for reformulating the user's informal specification into the particular form required, maintaining consistency, avoiding ambiguity, and ensuring completeness.

This basically standard language translation task is not difficult if the difference between the input and output languages is small and the input is understood. We have specifically designed our formal specification language to minimize the differences between it and the informal input to eliminate the first of these problems. Thus, the key issue is *understanding the input.*

GOALS AND APPROACH

Previous efforts have limited themselves to the use of formal specification languages, while precluding informal specification, precisely because of this difficulty in understanding such informal specifications. This limitation shifted the entire burden of creating and dealing with the formalism completely to the human specifier. We believe this extreme position is unwarranted and imposes an artificial barrier between computers and potential users. In a suitably constrained environment, much of the understanding of an informal input can be handled by a computer system that then undertakes the responsibility to re-express it in the required formalism.

How can such understanding be automated? First, we must define the nature of informal languages. The primary distinction between formal and informal language is that in the latter *information is suppressed*, yielding partial descriptions and constructs rather than complete ones. These partial descriptions and constructs result in ambiguity that can be resolved only by context.

But what information is suppressed? It is far from random. The originator of the informal communication suppresses that information he believes the recipient can correctly fill in and may suppress temporarily details not central to the main topic being communicated. Thus, informal languages are effective because they allow both the originator and the recipient to focus on the relevant issues and to suppress details that can be correctly inferred and that, if present, would detract from or blur the issues being highlighted.

Therefore, to automate the understanding of informal languages one must disambiguate partial descriptions and constructs by supplying the suppressed information, felt by the originator to be obvious from the context. Unfortunately, providing such disambiguation has been remarkably difficult because of the large variability in how context affects the disambiguation and the undetermined amount of unstated world knowledge that might bear upon the resolution.

On the other hand, considerable success can be gained by automating disambiguation in a highly constrained domain. Fortunately, programs are very highly constrained objects (one reason they are so hard to construct), and people seem to describe them in rather limited ways.

For the last few years, we have been building a prototype system that embodies considerable knowledge of how people describe programs and what makes such programs well formed. This special knowledge provides a powerful basis for disambiguating informal program specifications. Our prototype system has understood and correctly formalized several real-world, albeit simplified, specifications extracted from actual military MIL-SPEC 490-B5 specification manuals. The system has also successfully handled twenty five perturbations of these examples as a demonstration of its robustness on such small simplified specifications. We believe that these results demonstrate the basic feasibility of our approach.

PROGRESS

Having successfully demonstrated the prototype SAFE system on small simplified informal specifications and its robustness on perturbations, we turned our attention to the problems of handling larger realistic specifications. Handling such specifications is, we believe, the remaining watershed issue for our line of research. Unless such specifications could be handled, only a laboratory demonstration rather than a firm technological basis for practical formal specification formulation tools would have been provided.

It was immediately clear that the prototype system could not be scaled up to handle larger specifications. First, we, like all other major LISP-based projects, had run out of address space. The ad hoc measures adopted to complete our work on the perturbations couldn't be pushed any farther. Second, the basic paradigm of the prototype system was oriented toward small rather than large specifications. It was used on a simple three pass "compiler" model in which each phase completely processed the specifications before the next phase started. Thus, the entire specification had to be available before processing could begin, and since processing advanced in lockstep, detailed analysis of one portion could not be used to aid the analysis of another. Third, as the prototype system evolved, more and more knowledge was added through procedural embedding to the point that it had become exceedingly complex to further augment it. Thus, we have been accumulating programming knowledge faster than it could be added to the prototype system. Furthermore, the control structure of the system based on backtracking was very primitive, preventing parallel exploration of alternative formalizations. Finally, it was apparent that the formalization task was being complicated

by the need to intermix language translation. That is, while determining what the proper formalization of the informal specification was, the system also had to translate high level constructs into low level ones because we did not have a suitably high level formal language to use as the output of the prototype system.

For all these reasons, it was clear that it would be much more productive to build a completely new system rather than attempt to extend the existing prototype. These limitations led directly to the four tasks (explained below) undertaken in FY79.

ACCOMPLISHMENTS AND CURRENT STATUS

1. Disc Resident Associative Relational Data Base

In anticipation of a large address space implementation of Interlisp, we did not want to introduce temporary complicated partitioning mechanisms into our system. On the other hand, some method of avoiding this limitation had to be devised in the interim. Since the address space limitation mainly affects data (because of Interlisp's code overlay feature) and since the majority of our data is contained in our associative relational database, we decided to create a disc resident version of this database to circumvent the address space limitation. This buries the address space circumvention mechanism inside the database and enables us to avoid dealing with it within our prototype system.

2. Incremental Assimilation Paradigm

To overcome our paradigm lockstep limitations, we decided to use an incremental paradigm in which each portion of the informal specification was processed by all parts of the system before the next portion was examined. This change enables the assimilation of defined concepts and process refinements that characterize large specifications. Furthermore, it allows the system to process a large specification piece by piece, in a manner more closely modeling human understanding. Later input could resolve ambiguities or force reinterpretation of previous portions. This incremental processing also simplifies system development by providing a finer granularity of test points than the previous all-or-nothing paradigm. Finally, and most importantly, this paradigm shift opens the possibility of a whole new area of application of this research and is the basis of our future plans (see FUTURE WORK section).

This paradigm shift, while central both to processing large specifications and our future research, complicates the system by requiring it to deal with incomplete specifications. Previously, in dealing with informal specifications, the system had to handle only suppressed information, i.e., implicit information that the specifier believed could be deduced from the explicitly specified information, Now, it must also handle specifications in which necessary information is, as yet, unspecified. This means that it cannot completely analyze a partial specification but must be prepared to leave some ambiguities unresolved until further information is obtained.

To minimize these complications by retaining much of the structure of the current system, while incorporating the advantages of incremental processing, we decided that the input grain size should be a *single sentence* that would be processed more or less sequentially as a three-pass augmentation of the previously processed incomplete input.

3. HEARSAY III Framework

This need to extend the system to handle incomplete input, combined with the needs to simplify the addition of programming knowledge and to introduce a parallel search mechanism, led to a decision to rebuild the prototype within the framework of a modern knowledge-based Artificial Intelligence system, for only in such a framework would the necessary flexibility and growth capabilities exist.

After examining the available possibilities, we decided that the ARPA developed HEARSAY II system offered the best framework. Unfortunately, this system was specifically designed for the speech understanding task, and so we generalized its mechanisms and removed its task dependencies to produce a new framework, called HEARSAY III, in which to rebuild the SAFE system. Basically, the HEARSAY III system is a generalized production system in which autonomous clusters of expertise (knowledge sources) react to the activity of other experts on a public, structured blackboard and produce changes and augmentations to its contexts. These knowledge sources can either DEDUCE modifications where there is no ambiguity about how to process the structure on the blackboard or ASSUME modifications when uncertainty exists. These assumptions produce alternative versions of the blackboard that can be explored in parallel. As in any production-based system, deciding which of the computer's relevant productions will be allowed to run is a central issue. Here this contention exists both

between alternative versions of the blackboard and between knowledge sources within the same version. Rather than provide a built-in "scheduler" to resolve this problem, we decided that issue was complex enough, and the need for experiment so great, that the full flexibility of the HEARSAY paradigm itself should be available. We therefore added a second blackboard to be used by scheduling knowledge sources to determine how to allocate processing to the competing regular knowledge sources. These scheduling knowledge sources can invoke other waiting scheduling knowledge sources on waiting regular knowledge sources, thus allowing arbitrarily complex scheduling regimes to be built through this knowledge source mechanism. Naturally, a small kernel must exist to resolve conflicts at the top level of this scheduling hierarchy. This system will be described more fully in a forthcoming research report.

4. Formal High Level Specification Language

Finally, to eliminate the intrusion of language translation issues into our central task of formalizing informal specifications, we decided to create an appropriate formal specification language in which the high level constructs found in informal specifications had a corresponding formal equivalent. Before constructing such a language we felt obliged to determine the requirements it had to satisfy. This becomes a major effort and resulted in a quite novel set of requirements[1] presented at the Specifications of Reliable Software Conference. We are currently working on defining a formal specification language that satisfies these requirements.

FUTURE WORK

The major effort for FY80 is to scale the SAFE system within the framework described above so that it can handle real, unsimplified specifications of practical size. A specific example, the ARPANET Host-IMP Protocol, has been chosen because it is well-written, widely known, and representative of an important class of applications.

Specifically, we will start analyzing the Host-IMP Protocol specification in detail to determine what capabilities are required to understand this class of specifications. General knowledge about this class of system will be identified (for example, that a protocol is a set of rules for exchanging information between two or more processes and

---

[1] See Balzer, R., and N. Goldman, *Principles of Good Software Specification and their Implications for Specification Languages*, ISI/RR-80-86 (forthcoming).

that transmission errors can occur), and methods of incorporating it into the system devised. Deficiencies in the current knowledge base and/or the protocol specification will be identified. A plan will then be generated and implementation begun on overcoming these difficulties.

Also, through our development of a formal specification language we have particular notions about how specifications should be written. They center on the issues of separating functionality from representation and optimization, and on providing abstract yet process-oriented specifications. These notions must be explicated as a set of standards for SAFE specifications. Then the Host-IMP protocol must be rewritten to conform to these standards. In particular, it is currently filled with representation issues that must be removed.

As always, we measure progress and test system capabilities through particular examples. Therefore, we will generate a few simplified versions of the rewritten specifications, allowing us to consider these issues individually and work them as the implementation proceeds. The entire unsimplified rewritten Host-IMP Protocol (about 15 pages) will then be converted to a formal specification for the Host machine. Finally, we will create several perturbations of this specification to test the system's robustness to variability of large specifications.

Furthermore, we will provide a rudimentary capability to *exercise the formal specification.* This will enable a user to watch the formal specification operate on selected data to ensure that the formal specification produced by the system matches the user's intent. Such a capability is important because just as it is difficult for a user to write a formal specification directly, so is it difficult for him to directly (i.e., by inspection) understand the formal specification produced by the system. Instead, by observing the behavior of the formal specification on appropriate cases, the correspondence between intent and result can be ascertained. Together with a paraphrase capability (which might be added later), this would provide a quite complete capability for user understanding of a formal specification.

Such a capability to exercise a formal specification is important for a second reason. In addition to providing information concerning the correspondence between the formal specification and the user's intent, it also provides an assessment of whether the

specification meets the user's requirements. Short of producing a formal statement of these requirements (with all the attendant difficulties of such formalization) and verifying that the formal specification is consistent with these formal requirements, the only recourse is to make such an assessment by appropriately testing (i.e., observing the behavior of) the specification. Rome Air Development Center (RADC) is currently funding such an effort (testing specifications) at ISI as an outgrowth of the SAFE project.

These tasks, in FY80, will mark the end of the SAFE project. By then, we will have demonstrated that informal specifications of a practical military size and complexity can be reformulated, by a computer system, in a formal specification language. However, the SAFE system will still be very much a laboratory prototype. It will not yet have a user-interface, a paraphrase capability, a robustness to syntactic variability, or real-time responsiveness. Each of these deficiencies could be addressed as part of a development project to utilize the SAFE technology.

However, before such an effort is undertaken, a basic limitation of our specification paradigm must be removed. The current system assumes that a carefully thought out, but yet informal, specification has been created for input to the SAFE system. This paradigm is realistic--it characterizes current military practice as embodied in MIL-SPEC 490-B5 specifications. Thus, as we have argued throughout this project, such capability would be directly relevant to improving military software specifications.

Yet this paradigm fails to account for two critical facts. The first is that a considerable amount of effort has been expended in creating the informal specification. Just as SAFE has recognized the importance of automating the translation from informal to formal specification, so too should it recognize that considerable aid can be provided by interactively helping the user *formulate* the informal specification. That is, the system should help the user assemble coherent informal specifications from small individual fragments that get successively refined, elaborated, and possibly replaced or corrected. Although part of such help would be in simply advising the user of those portions that are incomplete and/or inconsistent, the main task would be in integrating the refined and elaborated fragments into a coherent whole.

Very closely associated with such a formulation capability is the second critical fact unaccounted for by the current SAFE paradigm--that maintenance dominates the

life-cycle costs of software. After an initial informal specification has been created, transformed into a formal specification, and implemented in a computer program, the resulting system is constantly being revised and extended as the end user gains operational experience with the system. Some of the revisions correct inconsistencies between the formal specification and the implementation. As software development methodology improves, such revisions should become less frequent and less important. The other modifications, those that change the formal specification, will, however, become more prevalent as user organizations adapt to the existence of the software systems and as these systems are integrated with each other.

As these modifications are made, both the formal specification and the implementation must be updated and kept consistent with each other. As a separate effort (Transformational Implementation), we are developing the technology to create implementations of formal specifications guaranteed to be consistent with those specifications and designed to be easily changed to reflect revisions of the formal specification.

A similar revision capability must also exist for the formal specification itself, so that as revisions are made to the informal specification they are incorporated into the existing formal specification. The critical fact here is that one does not change the informal specification and then retranslate it from scratch into a new formal specification. Instead, the change is explained as a revision or elaboration of the existing informal specification and integrated into the existing structure.

Thus, both the formulation and maintenance activities, which have been neglected by the current SAFE system, are based on the capability to incrementally modify and extend a specification and to tolerate, at least temporarily, errorful specifications. Creating such a capability represents a major departure from the current SAFE system in both the underlying technology and the areas of intended application. For this reason, we have recognized that the FY80 tasks represent the logical completion of the current SAFE paradigm based on presenting an entire specification all at once to the system.

The incremental processing paradigm introduced last year for handling large specifications is the foundation for a new paradigm, based on revision and elaboration, to be embodied in a project supplanting SAFE. This new project, Specification Formulation

and Maintenance, beginning in FY80, with tolerance of errors within the specification, will recast the technology developed by SAFE to understand and formalize informal specifications into a prototype system for incrementally constructing formal specifications from informal descriptions, revisions, and elaborations.

FY80 will be a transition year in which the current SAFE project is phased down while completing the tasks described above, and work is started on the new Specification Formulation and Maintenance Project. In this first year the deficiencies of the SAFE system for the new paradigm will be carefully analyzed, and a plan to correct them will be formulated. Also, a few examples will be chosen to exercise the new incremental capabilities but not otherwise stress the system.

In FY81 these incremental specification examples will be converted to coherent formal specifications as a demonstration of the new paradigm. In addition, we will begin integrating the system into a suitable environment so that we can start to get some experience dealing with real users. This involves three main tasks. The first is to incorporate a then-existing natural language interface for all interactions between the user and the system. The second is installing our own paraphrase capability to explain the formal specification produced and the rationale used by the system in arriving at the resulting specification. Finally, to effectively deal with such real users, we will choose a particular application area and begin incorporating the application expertise of that domain so that the system is knowledgeable in that area.

If these efforts to make the system accessible to outside users proceed well, we will plan for construction of a Prototype Specification Formulation and Maintenance system for test use by a selected set of military users.

5. NETWORK SECURE COMMUNICATION

Research Staff:
E. Randolph Cole
Stephen Casner
Eric Mader
Gertrud Mellstrom

Consultant:
Danny Cohen

Support Staff:
Mamie Chew
Robert Parker
George Dietrich
Orallo Garza

INTRODUCTION

Today prototype packet voice systems are in operation on the ARPANET, the SATNET, and the PRNET. The experience gained with these prototype systems, along with accompanying theoretical studies, indicates that integrated packet voice and data systems are likely to be an efficient and economical means of meeting future communications needs.

Since the ARPANET, SATNET, and PRNET all have a bandwidth on the order of thousands of bits per second, experiments in packet voice have thus far been limited to one or two channels. In addition, today's packet voice systems generally use a minicomputer for a network host and a high-speed array processor to perform the voice processing, both of which are expensive pieces of hardware. Past and present packet voice work has included design, implementation, and measurement of these limited systems for packet voice and their supporting concepts and protocols. Additional effort has gone into making the algorithms for voice bandwidth compression (mostly Linear Predictive Coding, or LPC) perform better in the presence of noise and making them capable of handling input from an ordinary telephone.

Two technological advances will soon allow packet voice to leave the laboratory prototype stage and advance into full-scale experiments with hundreds, perhaps thousands, of voice channels. Those two advances are wideband packet networks and LSI voice processing technology.

For the past year, ISI and other ARPA NSC contractors have been working on preparations for the Wideband Packet Satellite Experiment, which will begin in FY 80. The wideband (WB) packet satellite network will consist initially of two sites, ISI and Lincoln Laboratory (LL). Later two more sites, SRI International and the Defense Communications Engineering Center (DCEC), will be added. The WB satellite channel will

have a bandwidth of at least 1.5 megabits per second. Initial work will involve hardware and software installation and testing, followed by initial point-to-point packet voice experiments and soon after by implementation of packet voice systems capable of ten or more voice channels. Further work will expand the system to its design capacity of hundreds of voice channels.

Voice bandwidth compression work in the past year has included addition of a improved pitch tracker capable of handling telephone input to the real time LPC software for the Floating Point Systems AP-120B array processor. In addition, software was written for the FPS to implement the same LPC algorithm used by the Lincoln Laboratory LPCM vocoder. This capability will be used for initial internet packet voice experiments with ISI on the ARPANET and LL and others on the SATNET.

Useful packet voice systems for the future will also have to work in an environment of many interconnected networks. Therefore considerable effort in the past year has gone into development of an advanced network voice protocol, called NVP II, capable of operation in an internet environment.

PROBLEM BEING SOLVED

As more and more sophisticated technology for command and control is transferred out of the laboratory and into the field, the accompanying need for communications, particularly secure communications, becomes more severe. Voice, graphics, facsimile, and data must be transmitted quickly and securely. The ISI NSC group has been working to develop, implement, and test systems and methods for packet voice communications, and is beginning work on packet transmission of integrated multimedia communications. The results of this effort can readily be applied to meet the military's needs.

The ARPA NSC effort has achieved its first goal, which was to demonstrate a high-quality, reliable packet speech transmission system. This was done using relatively expensive minicomputers as network hosts and high-speed general-purpose signal processors for the necessary speech processing. Present efforts have concentrated and future efforts will concentrate on decreasing the size and cost of packet voice facilities and improving the transmitted voice quality.

As stated above, other problem areas include development of packet voice systems which use wideband networks to handle hundreds or thousands of users, and the extension of packet voice techniques into the internet environment. Both these areas need to be explored in order to build packet voice systems which will serve the large-scale communications needs and the diverse network environments the military is apt to experience in the future.

GOALS AND APPROACH

Currently there are at least four major goals for packet voice research and development:

- Development and testing of large-scale packet voice systems using wideband channels to accommodate hundreds or thousands of conversations,
- Extension of packet voice techniques into the internet environment, allowing a user on one network to communicate with a user on a different network, perhaps with several other networks in between,
- Reduction of the size, weight, and cost of packet voice terminals,
- Integration of other media, such as computer-generated graphics, with packet voice, forming a multimedia communications capability.

During the past year, the ISI NSC project has participated in preparations for the WB Packet Satellite Experiment, which will begin in FY 80. It is anticipated that the WB network will be the first with the capacity to support large-scale packet voice systems with as many as a thousand users. The WB experiment will require development of methods for interfacing large numbers of packet voice terminals to a single network node, as well as efficient multiplexing schemes to use the channel to its best advantage. The WB experiment will also provide a major opportunity to develop high-quality user-oriented conferencing systems. The high bandwidth will also allow future developments of powerful multimedia communications systems, with graphics, facsimile, text, data, and even bandwidth-compressed video in addition to voice.

Efforts to extend packet voice into the internet environment have been concentrated on development of an advanced network voice protocol, called NVP-II. The new protocol incorporates the experience gained with previous network voice protocols such as the

original NVP and its extension into conferencing, the NVCP. The NVP-II will use a lower-level protocol, called the Stream Protocol, or ST Protocol, to handle the actual data transmission. The ST Protocol was defined in the past year as a result of a need identified in the process of NVP-II development, and will provide a "guaranteed" bandwidth for packet voice and other media, as well as multiplexed and multiaddressed delivery in the internet environment. Several "strawman" specifications for NVP-II have been drawn up by ISI and others, and an implementable specification will be completed during this fiscal year.

Another objective of the ISI NSC project is to aid in the effort to transfer packet voice technology out of the laboratory into small, low-cost packet voice terminals. This task will require careful adaptation of present networking techniques and protocols if the result is to be usable with many different vocoders and networks, including an internet environment. ISI's role in this effort will be to provide protocol, networking, and user interfacing expertise.

The last of the project's four major goals is to integrate packet voice and other media, particularly computer-generated line-drawing graphics, into multimedia communications systems. The NVP-II will not be limited to transmission of only packet voice, and it is anticipated that graphics can be added fairly easily. This effort is in the system definition stage, and will be done in cooperation with the ISI Command and Control Graphics Project.

The primary approach of the ISI NSC project continues to be one of flexibility. Vocoder techniques are continuing to develop rapidly. New types of packet networks, such as the WB network, are appearing, and existing networks are evolving and being interconnected. Therefore research and development oriented packet voice systems and protocols must be kept as flexible, general, and expandable as possible within the real-time constraints of the task. For example, both the NVP and NVCP are as independent as possible of the vocoding techniques themselves. Parameters specific to the particular vocoder in use are confined to one module of the protocol; a new vocoder can be accommodated simply by changing a table. This philosophy will of course be continued in the design of NVP-II.

Flexibility is particularly necessary in testing and comparing alternative approaches to a single task. For example, users of packet voice systems should not be restricted to a single type of user interface, but should be able to pick the type of interface which best suits their needs. The NSC packet voice system can handle three types of user interface, ranging from a sophisticated CRT-based status display to a telephone interface which allows users to participate from an outside telephone via the pushbutton pad.

The ISI NSC project has always tried to maximize the usefulness of its work to the ultimate user. A system which is difficult to understand and use will not be accepted, no matter how carefully it is designed and implemented.

PROGRESS

Voice Messages

The concept of computer-delivered text messages is almost as old as the ARPANET itself. Beginning with simple programs merely to read individual messages in a message file, the state of the text message art has progressed to complex message handling systems which give the user powerful controls over every step of the process from generating a message at its source to filing it away by subject at its destination.

With the success of text message systems and packet voice, one can envision an even more powerful message system consisting of segments of packet voice along with or instead of the text. The voice messages could then be created, edited, transmitted, and read just as text messages are today (using different equipment, of course).

Two types of questions need to be resolved with regard to voice messages; the technical questions of storage format, transmission methods, etc., and the user-related questions of the role of voice messages, how to manipulate them, and indeed whether or not voice messages are useful at all. Questions of the first type, the easiest to answer, must be settled before experimental systems can be built to investigate the user-related issues.

The ISI NSC project had previously built a prototype voice message system based on the concept of *voice files*. This fairly simple voice message system used an extended Network Voice Conferencing Protocol (NVCP) for real-time storage of voice on disk files.

Using the slightly modified NVCP, the disk file appeared to be another vocoder, which could record or play back on demand. This initial voice message system allowed users who had already done real-time voice under NVCP to quickly try their hand at voice messages.

This first voice message system had its advantages and disadvantages. It was easy to implement and worked as well as the real-time packet voice system itself. Voice message files were stored at a central location. However, it required the voice to be transmitted in real time to and from the voice file. No editing was implemented. The system was not related to or compatible with standard network (text) message systems, or the proposed Internet Message Protocol [5].

During the past year, the ISI NSC project has proposed a model and a format for voice messages that are compatible with the Internet Message Protocol and takes full advantage of its many features. NSC Note 137 [2] describes the proposed format.

Some of the advantages of the proposed format for voice messages are:

- Real-time network response is no longer required,
- The format was designed for the Internet environment,
- Multiaddress capability is included,
- Multiplexing capability is included,
- Local editing is possible,
- Voice and text are integrated in the same message.

It should be noted, of course, that many of the above features are a result of the Internet Message Protocol and its capabilities. It should also be noted that [2] proposes a format for multimedia messages in general, not just voice along with text.

Internet Voice

From a beginning with the ARPANET just ten years ago, packet networks have proliferated rapidly, both in number and type. Major ARPA-sponsored networks include the ARPANET, the PRNET, and the SATNET, and the WB packet satellite network is in the construction stage. The existing networks are becoming more and more interconnected, and considerable Internet experimentation has been done.

It is therefore evident that for packet voice to be fully useful it must operate in an internetwork environment. Normal packet communications in an internet environment must be done using a higher-level Internet Protocol (IP). Local network headers are inserted in front of the IP header to transmit a packet across each particular local network.

The real-time requirement of packet voice communications makes the task of internetwork packet voice particularly difficult. Progress has been made in this area, however, with the definition of the Stream Protocol (ST) by a group of NSC contractors. The ST is an extension to the IP which creates an efficient multiway connection among all the networks involved for packet voice use. A "reserved," or "guaranteed," bandwidth on each network is obtained by the ST (as much as possible). Since on some packet networks, particularly packet satellite networks, such a pre-allocated bandwidth is called a *stream*, the new protocol is called the Stream Protocol. The accompanying section on protocol work explains the ST further. Much of the intelligence of the ST will be located in the gateways between networks, however.

The ISI NSC group has been involved in the past year with preparation for a preliminary internetwork speech experiment, which is currently in the initial testing stage. Sites in this preliminary experiment include Lincoln Laboratory, the Norwegian Defense Research Establishment (NDRE), and University College, London (UCL), all on the SATNET, and ISI on the ARPANET. A diagram of the experiment is shown in Figure 5.1. The purpose of the experiment is to demonstrate the feasibility of internet voice and allow the participants to gain experience with the internet environment for voice.

The preliminary experiment does not use the ST, since that protocol has not been completely specified yet, but rather uses extended code in the ARPANET-SATNET gateway at BBN. Existing voice protocols for the ARPANET and SATNET are used without change.

Preparatory work at ISI has included revision of the FPS-based real-time LPC vocoder to be compatible with the Lincoln Lab LPCM hardware LPC vocoders used in the SATNET packet voice experiments. Revisions were also made to ISI's real-time SPEECH conferencing program which runs on the ISI NSC PDP-11/45 host machine. These revisions included revising the bit packing and unpacking routines to handle the Lincoln format. A successful initial test of the system has just been completed, with all four

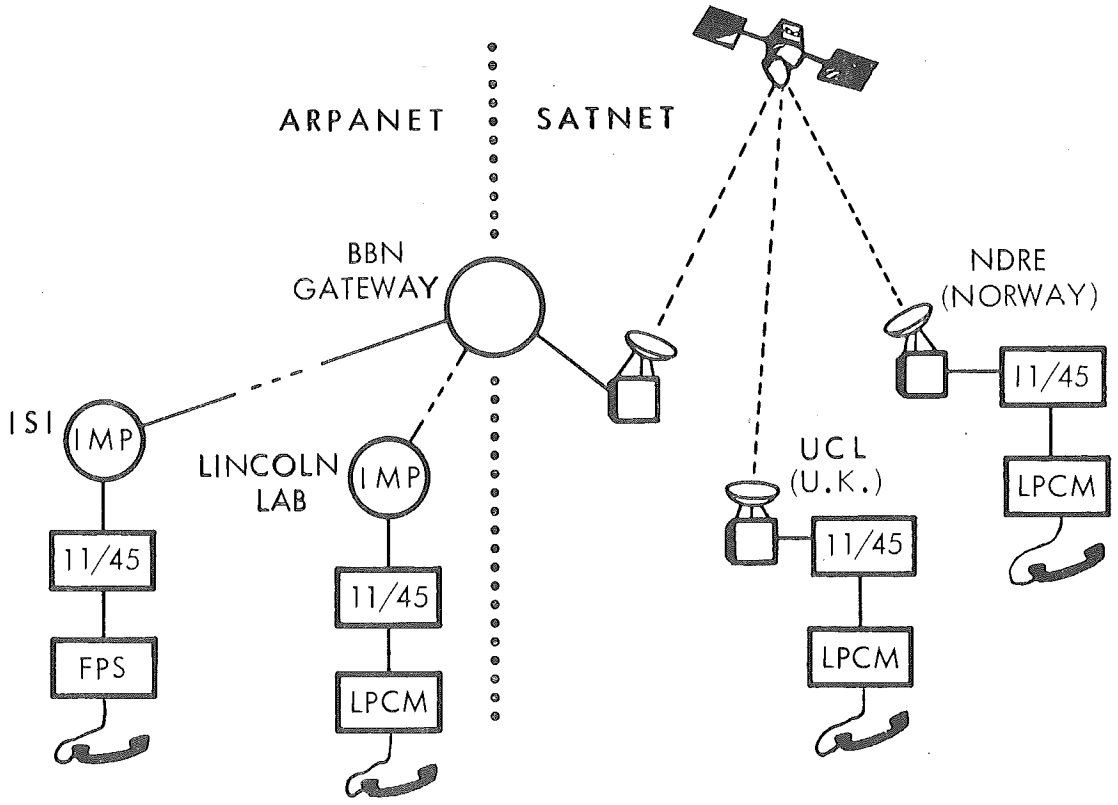


Figure 5.1. A diagram of the Internet Voice Experiment

sites (ISI, LL, NDRE, UCL) participating. Voice quality was quite good, generally indistinguishable from that of the ARPANET. The four-site system will be used to give a demonstration for the ARPA Internet group at their September meeting at UCL.

Cepstral Pitch Tracker

Most of the problems with modern narrowband vocoders involve the pitch tracker. Automatic tracking of human voice pitch is a difficult pattern recognition problem, and any degradation of the voice signal, whether noise or spectral distortion, makes the problem even more difficult. Unfortunately, the human ear is very sensitive to pitch errors. Considerable effort has gone into making pitch trackers as robust as possible, with success in some areas and not in others.

A common problem with many existing pitch tracking algorithms is that they cannot handle voice input from a conventional telephone. The SIFT (Simple Inverse Filter

Tracking), which the ISI NSC group has used in the various FPS-based real-time LPC algorithms, is one of those that cannot handle telephone input. The reason behind this problem is that the telephone system filters out frequency components in the voice below about 300 Hz. Many pitch tracking algorithms, including SIFT, utilize the normal voice's fundamental pitch frequency of from 50 to 350 Hz or so to determine the pitch. The human auditory system seems to be able to compensate for the loss of the fundamental probably by using the higher-frequency harmonics of the pitch, which are still present. If the fundamental pitch frequency of the voice is gone, a pitch tracking algorithm which depends on it will produce a poor, almost random, pitch output, very disconcerting to listen to and very difficult to understand. However, the telephone in its many forms is the most common voice input device around, and the usefulness of narrowband vocoders would be greatly limited if they could not handle voice input via a telephone (voice output via a telephone is not a problem). Narrowband vocoders will continue to be much more expensive than an ordinary telephone for a long time, and it will be desirable for several users to share a narrowband vocoder with access by telephone.

Some classes of pitch tracking algorithms, however, can operate without the existence of the fundamental pitch frequency. Some of these pitch trackers use pitch harmonics rather than the fundamental. A class of pitch tracking algorithms called *cepstral* pitch trackers makes use of the harmonic structure rather than the fundamental, and also seems to produce superior results with undergraded voice input.

The cepstrum of a segment of speech is calculated by transforming the speech using a Fourier transform, taking the log magnitude of the result, and inverse Fourier transforming the log magnitude. The word *cepstrum* is a transposition of *spectrum*, and the cepstrum has some properties like these of a frequency spectrum, and also some of the properties of the time domain. The particular feature of the cepstrum which makes it useful for pitch tracking is that it exhibits strong peaks corresponding to periodicities in the original time-domain waveform. A cepstral pitch tracker looks for the largest peak in the cepstrum, and uses the position of that peak as an initial estimate of the pitch period. The initial pitch estimate is compared to the pitch estimates from previous frames and a final pitch estimate generated.

In order to allow telephone input to its real-time LPC-based systems, ISI has implemented the Modified Cepstral Pitch Tracker, developed at SCRL by Markel [4]. The new pitch tracker required significant modification to ISI's real-time FPS software, since it is completely different from the SIFT pitch tracker.

A block diagram of the Modified Cepstral Pitch Tracker is shown in Figure 5.2. A 40-millisecond segment of input speech (with pitch period T) is first windowed with a Hamming window and transformed into its spectral, or frequency domain, representation via an FFT (Fast Fourier Transform). The log magnitude of the spectrum is taken and a spectral window is applied to it which deemphasizes the high-frequency components, which are often noisy. Then the windowed log spectrum is transformed using an inverse FFT, thus generating the cepstrum. The cepstrum is then weighted with a predetermined cepstral window, which increases with cepstral index, in order to weight all elements of the cepstrum equally. Finally a peak picking routine finds the cepstral peak (if one exists), and a pitch heuristic uses the current peak position and those from two previous frames to determine the final pitch estimate. The pitch heuristic also determines whether the speech is voiced or unvoiced.

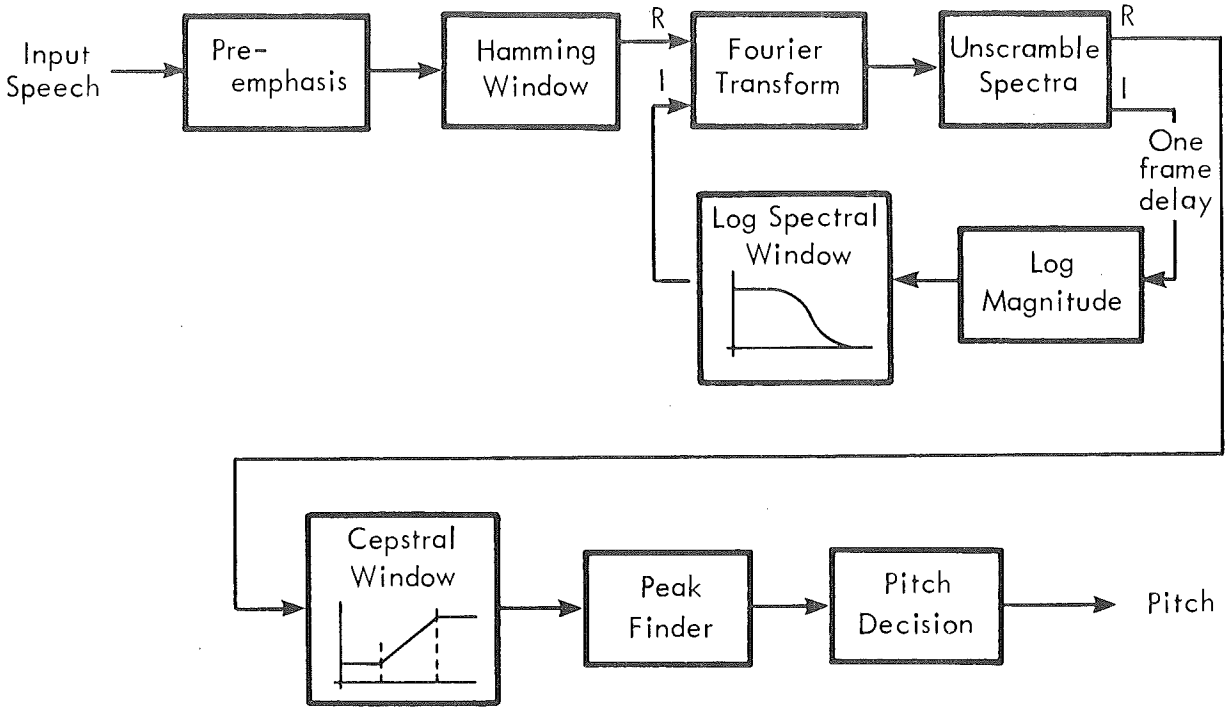


Figure 5.2. Block diagram of the Modified Cepstral Pitch Tracker

The new Modified Cepstral Pitch Tracker was implemented using standard FPS-supplied subroutines wherever possible. This should result in improved flexibility and maintainability. Runtime for the new pitch tracker is more than twice as long as that of the SIFT pitch tracker: 4.07 milliseconds per frame compared to 1.70 milliseconds per frame. The new pitch tracker required 243 more words of program memory. The LPC vocoder with the cepstral pitch tracker takes 62 percent of the available computing time compared with 37 percent required by the same LPC with the SIFT pitch tracker. An upcoming revision and cleanup of ISI's real-time LPC software will reduce the percent of the available time required by the cepstral pitch tracker by a considerable amount.

Performance of the LPC vocoder with the modified cepstral pitch tracker with telephone input is quite satisfactory. All cepstral pitch trackers seem to have trouble making an accurate voiced/unvoiced decision under some circumstances, and the modified cepstral tracker is no exception. An effort is being made to improve the accuracy of the voiced/unvoiced decision. This is a minor problem, and the overall performance of modified cepstral pitch tracker is very good.

Measurements

Last year's annual report [3] contained an extensive summary of the ISI NSC project's measurement of ARPANET packet voice transmissions along with results and conclusions. A few additional measurements have been made, with no new conclusions to report. The results of the measurements program were also summarized in [1] which NSC project members Steve Casner and Eric Mader presented at the 1978 National Telecommunications Conference held in Birmingham, Alabama,in December.

ISI and the other sites involved in planning the wideband packet voice experiment have included a significant measurements component in the experiment plan. It will be possible to collect measurements from any of the participating sites under control of any one of the other sites, without requiring human assistance at the other participating sites. Such a capability will provide more measurement data of higher quality than that which was possible on the ARPANET.

Additional Real-Time Capabilities

In addition to the Modified Cepstral Pitch Tracker, several additional capabilities have been added to the ISI real-time LPC software, including the implementation of the

algorithm used by the Lincoln Lab LPCM vocoder and modification of several of the existing fixed-rate LPC vocoder systems to run at a 50 frames per second rate. Emulation of the LPCM algorithm was necessary for ISI to participate in the preliminary Internet voice experiment described above, and the slower frame rate frees up FPS cycles for future additional processing such as noise reduction algorithms for high-noise environments.

Modifications to the real-time FPS LPC software to implement the LPCM algorithm included replacement of the normalized form synthesis filter with a two-multiplier form synthesis filter, replacement of the encoding and decoding table sets and the encoding software, and necessary changes to implement a 50 frames per second analysis rate. The resulting implementation gives good quality speech at a 2400 bits/second rate. The new LPCM software will also be used in the early stages of the wideband packet voice experiment, as described elsewhere in this report.

The ISI LPC vocoders were originally written to run at an analysis rate of 100 frames per second. This high frame rate was originally thought to produce better pitch in the synthesized voice. The so-called ARPA Phase II variable frame rate LPC vocoder developed by BBN needs to run at a 100 frames per second analysis rate in order to handle sharp voice transitions properly. However, it has been found that in the case of the ARPA Phase I fixed rate LPC vocoder, the LPCM implementation, and similar fixed-rate LPC vocoders, 50 frames per second is quite satisfactory. The slower frame rate requires less FPS cycles, roughly 4.5 milliseconds per 20 millisecond frame compared to 7.1 milliseconds per 20 millisecond frame previously (for Phase I LPC with the old SIFT pitch tracker). The reduction in run time will free FPS cycles to the addition of new capabilities to the ISI real-time LPC system, such as the University of Utah's SABRE noise reduction algorithm.

In the past year the ISI NSC group delivered its real-time EPOS operating system and its supporting software to the NSC group at the University of Utah and consulted in its installation. In addition, some effort was spent helping the NSC group at BBN debug the modifications required to run the real-time network voice software on the hardware configuration at BBN. The EPOS operating system and supporting software was previously installed at BBN.

The University of Utah NSC group has a PDP11/45 and an FPS AP-120B, and is now able to run the real-time LPC vocoder software originally written at ISI. This real-time capability should greatly enhance Utah's algorithm development work, since the effect of changes in the algorithms can be observed and measured over a much larger data base. The Utah NSC group is not involved in voice experiments over the network because their role is primarily one of developing and improving algorithms for speech compression and noise reduction.

The problems with the real-time network voice code at BBN were solely a result of different hardware. The BBN NSC group owns a PDP11/40, which differs from the PDP11/45 (like the one at ISI) in some very significant respects. In addition, BBN's A/D and D/A converters are on their old SPS-41, which is interfaced to the PDP11/40 via a shared memory, while ISI's converters are directly connected to the FPS. The different hardware systems caused the network voice code to act differently, causing pops and clicks in the output speech. The problem, once found, was fixed by changing the operating priority of one code segment.

WB Preparations

For the past year ISI has participated in preparations for the ARPA-DCA Wideband Packet Satellite experiment. The WB experiment will use ground stations at ISI, Lincoln Lab, SRI, and DCEC. ISI's participation in the planning has involved assistance in the preparation of the overall experiment plan and development of several experiments for which ISI will be primarily responsible, including a multi-line PCM system with access via the switched telephone network, and the investigation and development of a video capability. These experiments will be described in more detail in the Future Work section of this report.

The initial packet voice part of the Wideband experiment will consist of three phases. The first phase will consist of point-to-point voice communication between ISI, LL, and SRI, with probably only one voice terminal at each site. The second phase of the voice experiment will involve conferencing experiments with two or three voice terminals at each site. It is hoped that the third phase will see twenty or more voice terminals split between the three sites. The current target date for the first phase is September 1980, with the second and third phases currently planned for calendar 1981.

FUTURE WORK

With the exception of some work in the area of integrated voice/graphics conferencing, the ISI NSC project's future efforts will be concentrated on applications using the WB network, primarily including voice and low-bit-rate video. Initial voice work on the wideband network will be done in the following steps:

- An initial point-to-point (i.e., two-party) communications system between ISI and Lincoln Lab, using 2.4 kb/s LPCM. The purpose of this initial step is to test the WB network, the voice hosts, and the protocols used. The current goal for attaining this step is October 1980.

- A multi-party system, with two or more voice terminals at each of three sites: ISI, Lincoln Lab, and SRI. Other voice algorithms may be used in addition to 2.4 kb/s LPCM, such as 16 kb/s CVSD and 64 kb/s PCM. This capability is currently scheduled for February 1981.

- A multi-party conferencing system, with a total of twenty or more voice terminals at the above three sites. The exact hardware configuration has not been fixed, but a major portion of ISI's capability will be the multi-line PCM system described below. This task is currently scheduled to be done by October 1981.

These three steps are the initial part of the WB voice effort. Since the ultimate goal of the WB effort is to develop a system capable of hundreds or even thousands of voice connections, they are just the beginning. It is hoped, however, that by the time the multi-party conferencing system is done that the characteristics of the WB network will be fairly well understood, and the rest of the job will be largely an engineering task.

As a major part of ISI's WB voice effort, the ISI NSC project will build a multi-line PCM system which can be called from any pushbutton telephone. Conversely, the system will be able to dial out to any telephone. In this manner, a local access scheme (the telephone network) is provided without additional cost or effort.

ISI will assist other wideband network sites in the development of compatible facilities, allowing the wideband network to be tested with routine cross-country telephone calls. Continuous experimental use will provide a much larger data base than would be available from a system used only occasionally for demonstrations. More importantly, the large potential user community will help to keep a higher load on the system to exercise it more strenuously.

Since the PCM channels are timeshared among several users, the number of external telephone lines required is based on the maximum desired number of simultaneous connections and not directly on the size of the user community. The number of lines proposed for the PCM system is approximately 10. The resulting maximum bandwidth could therefore approach 640Kb/s.

The phone line interface will connect a mixture of ISI PBX and outside telephone lines to the PCM vocoders. The interface incorporates a 2/4-wire hybrid circuit to convert from the normal 2-wire telephone lines to 4-wire circuits with separate input and output. The function of the hybrid is to suppress the return of a signal being output from the PCM system back into the input (suppression of local echo). Echo is also induced at remote points in the 2-wire system which requires other measures for suppression. SRI is currently investigating the remote echo suppression problem, and the results of their research will be incorporated into ISI's system.

A second function of the phone line interface is to decode touch-tone (DTMF) tones in the signal from the phone line and translate them into control codes delivered to the processor. This allows a user of the system to specify connection through the wideband network by using the pushbuttons on his phone. Likewise, the phone line interface must be able to encode control codes from the processor into DTMF tones so that connection information from a remote site can be used to dial out to a local telephone. It is not necessary that the phone line interface detect whether or not the destination phone is answered (a difficult task) because the ring-back tone can be returned through the PCM channel to let the caller decide whether or not the call completed successfully.

Currently available PCM "codec" (coder-decoder) circuits provide A/D and D/A conversion plus the capability to time-division multiplex some number of channels onto a common, high-bandwidth serial line. The PCM system can therefore be divided into two pieces which may be physically separated by a reasonable distance.

Unlike voice, video is a new area of experimentation for ISI. Consequently, some initial setup will be required before experiments with the wideband network can begin. Video bandwidth compression techniques which are the product of many years of research at other organizations will be imported; it is not ISI's goal to develop new techniques.

All video cameras and monitors will use NTSC composite video so that relatively inexpensive standard equipment can be used. This choice is reasonable since the bandwidth limitations of the wideband network will be more restrictive than those imposed by the standard-quality equipment. Digitization will be done by a "frame memory" to which special-purpose bandwidth compression hardware can be attached. A similar frame memory at a destination site will allow slowing the frame rate by repeatedly displaying the same frame on the destination monitor.

NTSC composite video consists of 525 scan lines per frame and 30 frames per second. When sampled for full resolution (512 samples per scan line), 8 million samples per second are required. In order to avoid grey scale contouring, each sample must be quantized in 6 to 8 bits. The resulting bandwidth is 48 to 64 Mb/s. This bandwidth must be reduced by a factor of 24 to 48, or an average coding rate of .25 bits per sample, in order to use a maximum bandwidth of 1 to 2 Mb/s on the wideband network.

Image bandwidth compression algorithms suffer from the usual tradeoff of computation complexity, compression efficiency and resolution. It will be difficult to build compression hardware for a reasonable cost which operates in real time and still offers good resolution at the available bandwidth. In order to investigate the performance of various compression algorithms, the NSC project's FPS AP-120B signal processor will be used to simulate hardware implementations of the algorithms in non-real time.

Overall resolution of the destination image is the product of the horizontal, vertical, intensity/color and temporal resolutions. Different images require different components of the resolution to be emphasized. It is likely that multiple compression techniques will be desirable:

- High-resolution/low-frame-rate for transmitting images of documents or graphs,
- Low-resolution/higher-frame-rate for tracking the movement of a human speaker, for example.

It might be possible to implement bandwidth compression hardware which dynamically adjusts between these two extremes depending on the amount of motion in the picture. Adaptation to lower overall resolution may also be required to accommodate occasional increases in error rates in the satellite network.

Several steps are planned in the development of compression hardware, generally in order of increasing compression and increasing complexity:

1. Slow frame rate, full resolution: This requires no extra hardware, but requires 6 to 8 bits per pixel;

2. Differential PCM coding: Reduction to 3 bits per pixel;

3. One-bit dithering: This reduces the bandwidth to one bit per pixel using a relatively simple algorithm but the resulting spatial resolution is not very good;

4. Transform coding: A whole range of algorithms is available, offering bandwidths as low as 1/2 bit per pixel, but the computational requirements are large.

5. Inter-frame coding: This technique can be applied to successive raw or coded frames to remove frame-to-frame redundancy. A large amount of memory is required to buffer multiple frames.

Like multiple streams of high bandwidth voice, digital video at approximately 1Mb/s cannot be processed or even transmitted by a standard computer system. Therefore video packets will be delivered to the Packet Satellite Imp (PSAT) by the Voice Funnel (if possible) or by a special-purpose interface.

IMPACT

As packet voice technology becomes more mature, its potential impact is becoming clearer. Although much of the original impetus for the packet voice effort was its potential value as a relatively low-cost means for secure voice communication, the inherent efficiency of integrated packet-switched data and voice networks is becoming more and more apparent.

The ARPANET packet voice experiment is already beginning to affect military plans for future secure voice communications systems, and will greatly influence plans for command and control systems, particularly when packet-switched integrated multimedia communications techniques are developed.

At long last the efforts by the ARPA NSC group and others working on high-quality low-bandwidth speech communications are beginning to have an effect on the consumer market. Texas Instruments has introduced a low cost ($50) consumer product using a

single-chip LPC synthesizer. The intense competition in the consumer market is sure to bring down the price of narrowband vocoders, and thus allow narrowband packet voice communications at a much lower cost.

REFERENCES

1. Casner, S. L., E. R. Mader, and E. R. Cole, "Some Initial Measurements of ARPANET Packet Voice Transmission," in IEEE Publication No. 78CH1354-0 CSCB, pp. 12.2.1-12.2.5, December 1978. National Telecommunications Conference, Birmingham, Alabama.

2. Finn, G. G., and E. R. Mader, A suggestion for multi-media messages within the Internet environment. USC/Information Sciences Institute, NSC Note 137, June 1979.

3. ISI Research Staff, *1978 Annual Technical Report*. USC/Information Sciences Institute, SR-79-14.

4. Markel, J. D., Some Preliminary Notes on a Modified Cepstral Pitch Extraction System (MOCEP). Speech Communications Research Laboratory, NSC Note 111, June 1977.

5. Postel, J. B., Internet Message Protocol. USC/Information Sciences Institute, IEN 85, March 1979.

6. INTERNETWORK CONCEPTS

Research Staff:
Jon Postel
Danny Cohen
Carl Sunshine

Research Assistants:
Greg Finn
Alan Katz
Paul Mockapetris

Support Staff:
Mamie Chew
Linda Sato

PROBLEM BEING SOLVED

This project explores the design and analysis of computer-to-computer communication protocols in multinetwork systems. The project has four task areas: (1) Analysis, (2) Applications, (3) Design, and (4) Concepts. Protocol Analysis is concerned with the correctness of protocols, in particular Transmission Control Protocol (TCP). Protocol Applications is concerned with the development of demonstration internetwork applications, in particular a prototype computer message system. Protocol Design is concerned with the development of network and transport protocols, in particular the Internet Protocol (IP) and TCP. Protocol Concepts seeks new approaches in applying packet switching to communication problems.

GOALS AND APPROACH

The long-term goals of this research are to provide appropriate and effective designs for the primary user service applications in the internetwork communication environment, based on a set of host and gateway level protocols that provide the full range of service characteristics appropriate to a wide range of applications and that have been specified and analyzed to ensure their correct operation.

Our approach is to pursue in parallel the analysis, application, and design of protocols. The interaction of these activities provides valuable insights into problems and potential solutions.

PROGRESS

Protocol Analysis

We have identified several program and protocol analysis tools and techniques that show promise of being of assistance in the study of protocols. We will explore the value of these tools and techniques by applying them to a series of example protocols that incorporate features of the TCP.

Protocol Verification Issues and Purposes

Why is protocol verification special? What factors distinguish protocol verification from program verification?

- Timing dependencies:

  Protocols typically use timers to trigger actions, and there are often race conditions between the occurrence of an event and the firing of a timer. Few current verification techniques are able to model time-based processing or timeout-activated actions.

- Distribution:

  Protocols typically are implemented in modules located in independent distributed processors, that is, they execute in a truly asynchronous parallel manner. Normally there is no shared memory between the protocol modules. Verification of concurrent or parallel programs often focuses on the mutual exclusion issue. In protocols there is no mutual exclusion problem since there is no shared memory.

- Probabilistic:

  Protocols are usually designed to overcome some inherent unreliable or probabilistic behavior on the part of some element of the communication system. Program verification techniques do not attempt to treat probabilistic situations. They reason about things that are true or false, not about things that are usually true.

- Nonterminating:

  Protocols are usually designed to execute forever. Protocol modules accept some input, run through some states in the process of delivering the data, and return to some waiting state (possibly with different values of some internal variables). Program verification considers the termination of a program an important verification condition.

What do we want to know about a protocol?

- "Correctness"

  We want to know if the protocol is "correct". We want to know if it does what we intended it to do.

- Progress

  We want to know if a communication process makes progress, that is, we want to know that messages do get delivered.

- Performance

  We want to know how well a protocol performs in terms of both external and internal measures. We want to know that parameters (e.g., message size, timeout period) are set to optimal values.

- Efficiency

  We want to know that the protocol meets some external measure such as delay or throughput.

- Resource utilization

  We want to know that the protocol meets some internal measure such as cpu utilization or memory utilization.

Protocol Applications

We produced an initial design for an internet message system [4] and, after a period of review by the ARPA research community, a revised edition of this design [5]. This design focuses on the communication of messages in a machine-oriented internal representation that provides for carrying data of several media including text, voice, facsimile, and graphics.

Significant progress was made on implementing a prototype internet message system that follows this design.

An Internetwork Message Structure

The proposed system provides for transmitting messages composed of a rigidly defined and structured set of command information fields and an arbitrarily defined and structured message content. The content may include text, facsimile, graphics, or voice data.

System Organization

This message system model takes the view that the message service can be divided into two activities: message reading and composition, and message delivery. The message reading and composition is an interactive

activity in conjunction with a User Interface Process (UIP). The message delivery activity is carried out by background processes called Message Processing Modules (MPMs).

The internetwork message system is concerned with the delivery of messages between MPMs throughout an interconnected system of networks. It is assumed that many types of UIPs will exist. The MPMs exchange messages by establishing full duplex communication and sending the messages in a fixed format. The MPMs may also communicate other information by means of commands.

A message is formed by a user interacting with a UIP. The user may utilize several commands to create various fields of the message and may invoke an editor program to correct or format some or all of the message. Once the user is satisfied with the message it is "sent" by placing it in a data structure shared with the MPM.

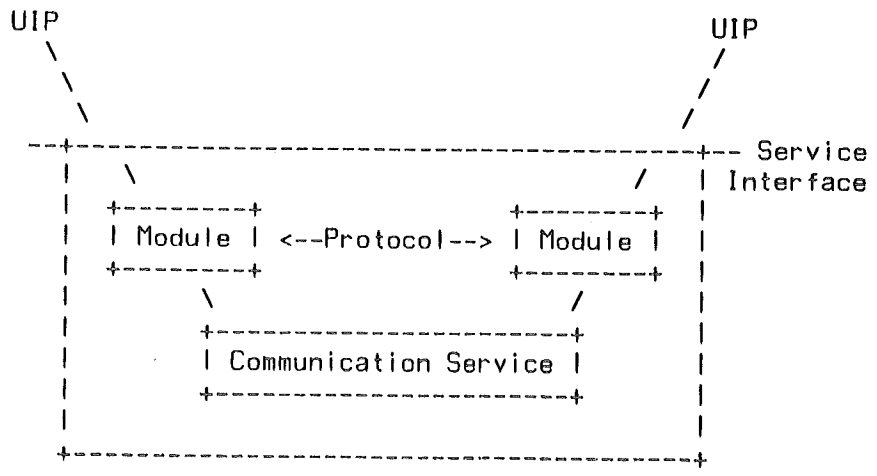
The MPM discovers the unprocessed input data (either by a specific request or by a general background search), examines it, and--using routing tables--determines which outgoing link to use. The destination may be another user on this host, a user on another host in this network, or a user in another network.

An MPM might know a way to establish direct connections to each of a few MPMs in other nearby networks, and send all other messages to a particular "big brother" MPM that has a wider knowledge of the internet environment.

The MPM calls on a reliable communication procedure to communicate with other MPMs. In most cases, this is a Transport Level protocol such as the TCP. The interface to such a procedure conventionally provides calls to open and close connections, calls to send and receive data on a connection, and some means to signal and be notified of special conditions (i.e., interrupts).

The MPM receives input and produces output through data structures that are produced and consumed respectively by user interface (or other) programs.

Several aspects of a distributed service have to be specified. First, there is the service to be provided, that is, the characteristics of the service as seen by its users. Second, there is the service it uses, that is, the characteristics it assumes to be provided by some lower level service. And, third, there is the protocol used between the modules of the distributed service.



Message Service

The User/Message Service Interface

The message delivery system accepts messages conforming to a specified format, attempts to deliver those messages, and reports on the success or failure of the delivery attempt.

The Message/Communication Service Interface

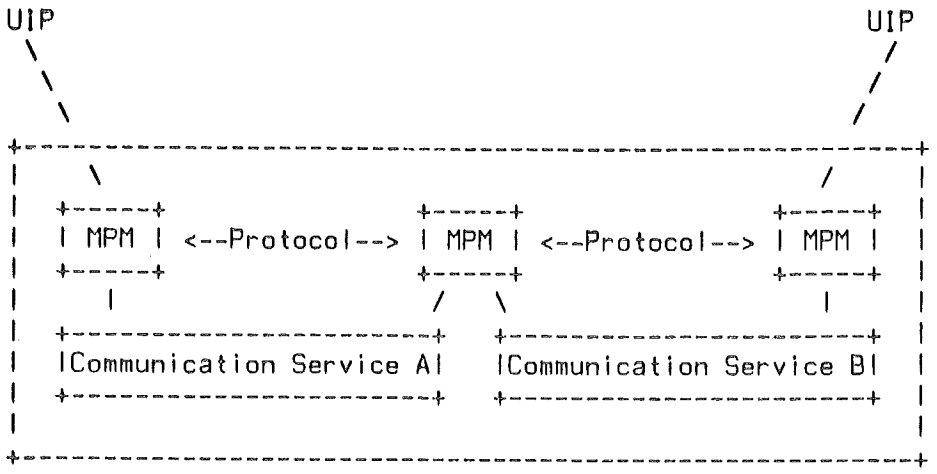
The message delivery system calls on a communication service to transfer information from one MPM to another. There may be different communication services used between different pairs of MPMs.

It is assumed that the communication service provides a reliable two-way data stream. Such a data stream can usually be obtained in computer networks using the transport level protocol, for example, TCP.

The Message-Message Protocol

The protocol used between the distributed modules of the message delivery system, the MPMs, is a small set of commands conveying requests and replies, encoded in a highly structured and rigidly specified format.

A pair of MPMs that can communicate reside in a common interprocess communication environment. An MPM might exist in two (or more) interprocess communication environments, and such an MPM might act to relay messages between MPMs in the environments.



Message Service with Internal Relaying

Message Structure

A message has three parts: the identification, the command, and the document. Each part is in turn composed of message objects. The identification part is composed of a transaction number assigned by the originating MPM, and the internet host number of that MPM. The command part is composed of an operation code, an argument list, the destination mailbox, and a trace, which is a list of the MPMs that have handled this message. The document part is composed of a header and a body. The message delivery system does not depend on the contents of the document part.

Commands

Only a few commands are defined:

Deliver:	Deliver the associated document to the address in the mailbox.
Acknowledge:	The message with the identifier in the arguments was delivered with the outcome indicated in the arguments.
Probe:	Verify that the mailbox is a valid destination.
Response:	Confirms (or denies) the validity of the mailbox in the argument.

Documents

The document consists of a header and a body. The header is a set of fields and their values. The body is a list of arbitrary data elements. In most cases, the body will be a series of text strings.

The header consists of a set of named items, each having a text string value. The following is the standard header:

Name	example value
DATE	1979-04-15-14:03-08:00
SENDER	Mamie@ISIE
FROM	Jon Postel <POSTEL@ISIB>
TO	Dave Crocker <DCrocker.of.UDEE@FWDR>
CC	Danny@ISIB
SUBJECT	Our Meeting

These are the basic header items; in addition, a variety of other header items can be used.

Elements

Messages are encoded into a set of typed data elements. The following elements are defined:

BOOLEAN	The Boolean element encodes the values TRUE and FALSE.
INDEX	Index encodes a small nonnegative integer number.
INTEGER	Integer encodes a twos-complement integer.
BITSTR	Bit String encodes arbitrary binary data.
TEXT	Text Strings encode ASCII characters.

These basic elements may be combined with two structuring elements:

LIST	Elements may be formed into lists, which may include lists.
PROPLIST	A property list of name-value pairs may be created.

Scenario

The scenario for a message transmission begins with a user interacting with a UIP to compose a message, an activity which may involve the use of a text editor. When the user is satisfied with the composition of both the message body and the message header, the user tells the UIP to "send" the message.

To send a message, the UIP formats the message into a message data structure of objects and elements and presents it to an MPM.

Upon receipt of a message the MPM examines the command section to determine the action to be taken. In many cases, the message is to be delivered to another MPM, specified in the mailbox.

To deliver a message to another MPM, an MPM consults a routing table and forwards the message along the path indicated by the table.

When an MPM receives a message and determines from the mailbox that it is the destination MPM, it delivers the document to the UIP, and forms an acknowledgment message addressed to the originating MPM.

When the source and destination MPMs are (logically) in direct communication (i.e., use the same communication environment), the delivery of a message will simply be the exchange of a "Deliver" and an "Acknowledge" command.

When the destination is not in the same communication environment as the source, the message must be relayed via intermediate MPMs. Generally speaking, the source MPM sends a "Deliver" command to the intermediate MPM. The intermediate MPM adds its identity to the trace and forwards the command to (or toward) the destination. The destination MPM delivers the message and sends an "Acknowledge" command to the message originating MPM via the intermediate MPMs.

Summary

The outlines of an Internetwork computer message system have been presented. The system uses type-encoded data elements to represent message objects organized into list structures. Messages formatted in this way may be transmitted between MPMs using a variety of underlying interprocess communication systems.

It is claimed that the typed and structured format is better suited for the provision of message capabilities beyond plain text. The inclusion of Facsimile, Graphics, and Speech information in messages will be important in the near future. This system provides a framework for the development of advanced message system features such as enciphering, accounting, and routing.

Message Addressing Transition Plan

We produced a plan for moving the ARPANET message system into the Internet environment [6, 7].

The discussion of a transition from the current ARPANET message environment and mechanisms to a more general Internet environment and richer mechanisms must consider techniques for continued activity during the transition. In addition, there is a current need for a mechanism to support the interaction of the several already existing NSW-like message environments with the ARPANET message environment.

The following goals should be addressed by the solution:

1. Minimum change to existing software.

2. Maximum user acceptance.

3. Maximum compatibility with the future internet message environment.

4. Minimum special transition software.

These goals are to some degree incompatible, so the evaluation should be expected to involve a tradeoff.

A crude model of the current situation and mechanisms of the ARPANET message environment follows. It is assumed the reader understands it well enough to dispense with a long description of how a message gets from A to B. It is important to note the types of processes involved.

In general there are several message composition programs (e.g., Hermes, SNDMSG) for each type of operating system or host in the network, as well as mailers, message servers (i.e., FTP servers) that receive the messages coming into a host and deposit them in mailboxes. In general there are several message processing (or reading) programs (e.g., Hermes, MSG, RD) for each type of operating system or host in the network. More developed programs are for both reading and sending messages.

Messages are transmitted as a character string to an address specified "outside" the message. The destination host ("YYY") is specified to the sending (or user) FTP as the argument of the "open connection" command, and the destination user ("XXX") is specified to the receiving (or server) FTP as the argument of the "MAIL" (or "MLFL") command. In TENEX, when a message is queued this outside information is saved in the file name ("[---].XXX@YYY").

This proposal suggests that messages for users in another network be sent to distinct per user mailbox names on a forwarding host; however, these mailbox names would have a common "network" part and a unique "user" part. By recognizing the common part the FTP server would put the message and the mailbox name into a single file to be examined by a routing daemon process. The routing daemon process would use mailbox name information to determine the actual destination.

Format:

Outside: [---].NSW-Joe@FWDR

Inside: To: NSW-Joe@FWDR From: Sam@ISIB

It is particularly instructive to work through examples with a mixture of mailbox destinations in the ARPANET and other networks in each of the "To:", "CC:", "From:", and "Sender:" fields and to see what happens when one wants to send an answer to all, just the "To:" or just the "CC:", or just the "From:" or "Sender:" mailboxes.

It is easier to talk about these things using examples. In the following, "NSW" is an example of a network name. "FWDR" is a host name or nickname for the forwarding host. For all of these solutions it is assumed that host tables can have alternate or nicknames for hosts, e.g., FWDR could map to 86 while ISI also maps to 86, although this is not essential. In addition, this solution provides a single forwarding point from the ARPANET into the destination net.

Also note that the information shown as the "outside" information is the TENEX representation. The key factor is that the mailbox argument value passed to the FTP server is the one in the string "[---].XXX@YYY", not anything from the header. Only the string "XXX" is passed to the FTP server.

Example:

Outside: [---].NSW-Joe@NSW

Inside: To: NSW-Joe@nsw, Bill@ISIA, NSW-Fred@NSW CC: NSW-Mike@nsw, NSW-Paul@nsw, John@ISIB From: Sam@ISIB

No changes are needed in message composing or processing programs. The FTP server has to put all the NSW-x users' messages into a single file that the routing daemon examines. The FTP server can do this on the recognition of the "NSW-" prefix without knowing all the legal individual users. In addition the FTP server puts the mailbox argument into the file with the message. This is necessary to avoid the loss of the "outside" information. The routing daemon can then look at the mailbox argument to determine where to forward the messages. It need not look at the inside of the message at all. The "answer" command works fine.

A problem arises about acknowledgments of message receipt. First the normal ARPANET message delivery mechanisms will say the message is delivered when the FTP server puts the messages in the file for the routing daemon to examine. However, if the routing daemon discovers a message is to be forwarded to a nonexistent user, the daemon can easily tell the original sender the exact destination user that is unreachable.

The only change needed in the software of the ARPANET message system is the FTP server at the FWDR host and the creation of the forwarding daemon.

Conclusions:

> This solution is based on the use of strictly "outside" information. Note that the existing ARPANET message DELIVERY system is based strictly on the use of "outside" information only. Also note that the problems that keep coming up in ARPANET message processing and composition programs have to do with the different possibilities for syntax (and semantics) of the "inside" information.

Protocol Design

We contributed to and edited two editions of the Internet Protocol (IP) and the Transmission Control Protocol (TCP) Specifications [8, 3, 9, 10]. The IP is a datagram-style gateway-level protocol that provides the addressing, routing, and fragmentation/reassembly functions in the internetwork. The TCP is a connection-style host-level protocol that provides end-to-end reliable ordered delivery of streams of data. These specifications were widely circulated in the DoD as the basis for a DoD internetwork protocol standard.

We wrote Internet Experiment Notes (IENs) and made presentations at Internet Working Group meetings on a number of protocol design topics that include: addressing, routing, type of service, multiplexing, user datagrams, and name servers [11, 32, 12, 13, 14, 1, 2, 15, 16, 17].

We participated in the Internet Working Group and provided support for the group's activities by producing meeting agendas and minutes [18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30].

Internetwork Protocols

The motivations for constructing computer communication networks--data and program exchange and sharing, remote access, etc.--are also motivations for interconnecting networks. This follows from the observation that the power of a communication system is related to the number of potential participants.

Interprocess Communication

When discussing computer communication, it is useful to recall that the communication takes place at the request and agreement of processes. Processes are the actors, the senders and receivers of data, in the computer communication environment. The effect of the layers of protocol built up in constructing the communications capability is to provide an interprocess communication system.

When a new host computer is to be connected to an existing network, it must implement the protocol layers necessary to match the existing protocol used in the network. The new host must join the network-wide interprocess communication system, so the processes in that host can communicate with processes in other hosts in the network.

The interconnection of networks requires that all the processes in the hosts of the interconnected networks have a common interprocess communication system.

Datagrams and Circuits

Two types of service are commonly discussed as appropriate for the network-provided interprocess communication service: datagrams and virtual circuits.

Datagrams are one-shot simple messages. Datagrams are inherently unreliable, since they are one-way and are not acknowledged, and messages may arrive in a different order than sent. Datagrams are simple to implement since they do not require the networks or gateways to record and update state information. Datagrams must carry complete address information in each message. A user transmits datagrams via send and receive actions.

Virtual circuits (or connections) are designed to be reliable and to deliver data in the order sent. Implementation of virtual circuits is complicated by the need for the networks or gateways to record and update state information.

Virtual circuits are created through an exchange of messages to set up the circuit; when use terminates, an exchange of messages tears down the circuit. During the data transmission phase, a short form address or circuit identifier may be used in place of the actual address. The user of a virtual

circuit must perform actions to cause the virtual circuit to be created (call set up) and terminated, as well as to send and receive data.

Each of these services is needed in a general-purpose communication environment. Datagrams are most efficient for transaction type of information requests such as directory assistance or weather reports. Virtual circuits are useful for terminal access to interactive computer system or file transfer between computers.

Gateways

Two or more networks are connected via a device (or pair of devices) called a gateway. Such a device may appear to each network as simply a host on that network.

Some gateways simply read messages from one network (removing that network's packaging from the messages), compute a routing function, and send the messages into another network (wrapping them in that network's packaging).

Since the networks involved may be implemented using different media, such as leased lines or radio transmission, this type of gateway is called a media-conversion gateway.

Other gateways may translate the protocol used in one network to that used in another network by replacing messages received from one network with different messages having the same protocol semantics sent into another network. This type of gateway is called a protocol-conversion gateway.

It should be clear that the distinction between media conversion and protocol translation is one of degree; the media conversion gateways bridge the gap between differing link and physical level protocols, while protocol translation gateways bridge the gap between differing network and higher level protocols.

The translation approach to network interconnection raises several issues. The likelihood of success in protocol translation seems negatively correlated with the protocol level. At the lower levels, protocol translation causes no problems. That is because the physical level and link levels are hop-by-hop in nature (though it should be noted that different protocols even at these low levels may have impact on the reliability, throughput, and delay characteristics of the total communication system).

At the network and transport levels, the issues of message size, addressing, and flow control become critical.

One must provide for message fragmentation and reassembly unless the maximum message size is limited to the size that can be transmitted on the network having the smallest limit.

The translation of addresses is a difficult problem when one network or transport level protocol provides a larger address space than the corresponding protocol to be translated to.

When end-to-end flow control mechanisms are used, as they commonly are in transport level protocols, difficulties arise when the units controlled are different--for example, when one protocol controls octets and the corresponding protocol controls letters.

At higher levels, the problems are more difficult because of the increased state information kept and the decreased likelihood of one-to-one translation of individual protocol messages.

Another difficulty is that each level further multiplexes the communication so that each connection, stream, channel, or virtual circuit must be separately translated.

Gateways may be thought of as having a "half" for each network they interconnect. One could model the operation of a gateway as having each gateway-half contain procedures to convert from a network-specific protocol into a standard protocol and vice versa.

Interconnection of Networks

The ARPA-sponsored research on interconnections of networks has led to a two-level protocol, the Internet Protocol and a Transmission Control Protocol. The IP is a hop-by-hop datagram protocol. The TCP is an end-to-end logical connection protocol. The model is that networks are interconnected via a gateway.

The IP header carries the information to deliver a message to a destination in the internet. The IP is implemented in the gateways and in hosts. A sending host prepares a message with an IP header and then selects a gateway in its own net to route the message. The sending host then sends the message with its IP header and a local net header to that gateway.

A gateway receives a message from one of the local networks it is attached to and strips off the local header. The gateway examines the IP header and determines the next gateway (or destination host) address in one of the networks it is directly connected to. The gateway then sends the message with its IP header a new local net header for that gateway (or host).

The IP has no provision for flow control or error control on the data portion of the message (the IP headers are checksummed). There are no acknowledgments of IP messages. This allows the IP to be very simple and the gateway implemented in small machines. A key point is that a gateway

has no state information to record about a message. At the IP level, there are no connections or virtual circuits.

A logical connection protocol, TCP uses end-to-end mechanisms to ensure reliable ordered delivery of data. It uses flow control, positive acknowledgments with time out and retransmission, sequence numbers, etc., to achieve these goals.

Addressing

The address is a fixed-size, hierarchically assigned global address.

Routing

Normally, the user has no influence over the route used between the gateways. The route may vary from one message to the next. No state information is kept in the gateways.

A user can insert a source routing option in the IP header to cause that particular message to be routed through specific gateways.

Buffering and Flow Control

The gateways do not control the flow-on connection, for they are unaware of connections or any relation between one message and the next message. The gateways may protect themselves against congestion by dropping messages. When a gateway drops a message because of congestion, it reports this fact to the source of the message. The TCP uses end-to-end flow control using windows on a per logical connection basis.

Acknowledgment

The IP has no provision for acknowledgments. The TCP uses acknowledgments for both error control and flow control. The TCP acknowledgments are not directly available to the user.

Recovery

Errors in a network or gateway result in a message being dropped, and the sender may or may not be notified. This inherent unreliability in the IP level allows it to be simple and requires the end-to-end use of a reliable protocol. TCP provides the reliable end-to-end functions to recover from any lost messages. The TCP uses a positive acknowledgment, time out, and retransmission scheme to ensure delivery of all data. Each message is covered by an end-to-end checksum. Because of the potential for alternate routing, the end-to-end communication may be able to continue despite the failure of a gateway.

Security

The IP provides an option to carry the security, precedence, and user group information compatible with AUTODIN II. Each network must enforce these parameters, and only AUTODIN II is prepared to do so. The TCP end-to-end checksum covers all the address information (source and destination network, host, protocol, and port), so if the checksum test is successful the address fields have not been corrupted.

Header Structure

The IP header is 20 octets (plus options, if used), but there is no call set up and no gateway state information. Thus, at the IP level, the header size vs state information trade off has been made toward large header and little (no) state. The TCP header is 20 octets (plus options, if used). There is a connection establishment procedure called the "three-way handshake," and significant state information is kept. In this case, there are both large headers and large state.

Summary

The ARPA networks are interconnected by using a common datagram protocol to provide addressing (and thus routing) information, and an end-to-end transport protocol to provide reliable sequenced data connections.

Protocol Concepts

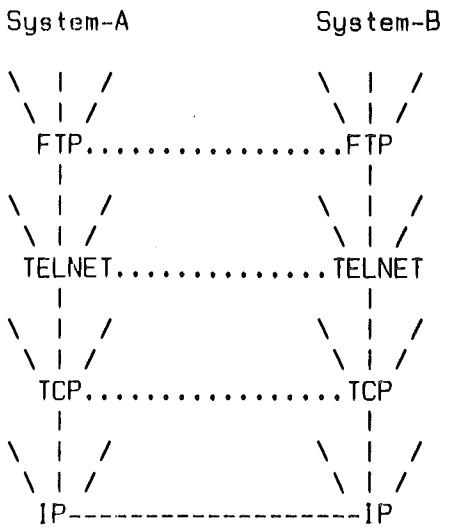
We have contributed to general planning of the internet communication system. In particular, we have identified the issues of nesting and combining as central to efficient multiplexing of messages.

On Protocol Multiplexing

Our thesis is that combining has the potential of achieving improved communication efficiency and that nesting determines the intrinsic layers of protocols, and these interact to impose an implementation style by forcing certain symmetries in protocols.

Protocol Structure

It is now a well-established practice of protocol design for protocols to be layered and symmetric in structure. The typical structure is shown in the following example, indicating the relation between the various levels of protocol, where those that appear above the others are considered as being of higher level.



This modularity results from the ability of each layer to accommodate several types and instances of the higher level; e.g., a single IP can serve several simultaneous instances of TCP (even in different machines). Each TCP can support several parallel instances of different user-service protocols (e.g., TELNET, FTP). At the top, TELNET, FTP, MAIL, and NGP have the ability to serve several processes.

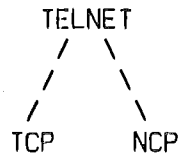
This modular hierarchy is called nesting and allows each level to multiplex several, possibly different, higher level protocols.

The source side plays the role of a funnel in which each level nests the higher level message inside its own level's message as data, and the destination side plays the role of a parser where each level passes the data of its level message to the appropriate higher level.

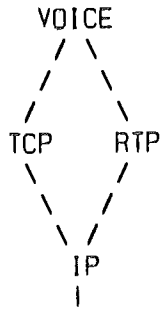
The information used at all levels of this parser is part of the total address of the terminal destination.

Note that there is also the possibility for multiple protocols at a lower level supporting a single protocol at a higher level, with the higher level protocol able to choose between the lower level protocols.

For example, TELNET could choose to use as its supporting transport protocol either NCP or TCP, since they provide equivalent functions and belong to the same level.



Another example is a voice protocol that might send control information via a reliable protocol such as TCP and the actual voice data via a real time protocol (RTP). Both TCP and RTP might call on a common lower level IP to transmit the data.



Nesting

The communication process between end users can be viewed as a series of steps, each of which packages (wraps) the essential information to be communicated in several layers of protocols. Each layer of packaging added in the transmission process has to be removed at a certain stage in the reception process.

The idea of nesting is that a message body may be another full-fledged message itself. We can formally define nesting as:

```
<MESSAGE> ::= <HEADER> <BODY>
<BODY>    ::= <DATA> | <MESSAGE>
```

For example,

```
1)  <MESSAGE> ::= <HEADER> <BODY>
2)  <MESSAGE> ::= <HEADER> <HEADER> <BODY>
3)  <MESSAGE> ::= <HEADER> <HEADER> <HEADER> <BODY>
                     |        |        |       |
                  (HEADER-1 (HEADER-2 (HEADER-3 (BODY))))
```

In particular, a TCP segment sent on the ARPANET has:

Header-1 = Local ARPANET Header
Header-2 = IP Header
Header-3 = TCP Header

The effect of this nesting in implementation is often that each unit of data released by a high-level protocol is treated independently asit progresses through the lower levels and accumulates more and more wrappings. This means that if two or more instances of high-level protocols are communicating with destinations close to each other and release data at essentially the same time, those data units are transmitted independently.

The independent transmission of data units to closely related destinations increases the competition for network resources.

It is possible to provide a protocol service, say TCP, to several simultaneous users in two ways: (a) by having a single TCP process timeshared (i.e., multiplexed) among them or (b) by giving each user an instance of TCP.

The choice between these two options must not be a part of the protocol specification, and should be left open to allow optimal implementations appropriate to the host environment. However, it is important that these two schools of implementation be compatible.

The inclusion of multiplexing makes this problem more explicit, since it addresses the issue of how and where multiplexing is done. In general, the inclusion of multiplexing should not exclude either the (a) or (b) implementation option, or their compatibility.

Combining

In recent years, with the rapid advances of Packet Switching technology, it became apparent that the marginal communication cost is low for bits, but high for messages (packets). Typically, this is reflected in the "per-packet" pricing policy, as opposed to a "per-bit" policy. This pricing policy represents a strong economical argument against small messages and suggests that messages should be made as long as practically possible. The argument in favor of the short messages is the requirement not to exceed certain delay constraints, which prohibits waiting until messages are filled.

In order to enjoy the best of both worlds, namely to have long messages without having to wait to fill them by individual users, combining is introduced. Combining is a simple scheme of sharing messages between several users communicating between the same systems, at some level. If combining is cleverly implemented, messages addressed to destinations that are "close enough" can be put together even though their end addresses are not identical.

Of course, messages may only be combined when they have the same transmission requirements, that is, when they demand the same type of service from the transmission system.

Combining attempts to reduce the number of messages (packets) communicated, but not necessarily the number of bits. Combining saves a few bits by taking advantage of the common portions of the headers (and addresses) of the individual messages and adds a few bits required in order to separate the combined message into the individual ones. The exact trade off in bits between the saving and the addition depends on implementation implementation but is not very significant. On the other hand, every piece combined is a message saved, which is significant.

A formal but simple way to describe combining is to treat messages as being composed of HEADER and BODY.

As before, we have:

```
<MESSAGE> ::= <HEADER> <BODY>
```

We want to send several data items with one overall header, for example:

```
<MESSAGE> ::= <HEADER> <BODY> <BODY> ... <BODY>
```

A cleaner way to specify this is:

```
<MESSAGE> ::= <HEADER> <BODY>
<BODY>    ::= <DATA> | <BODY> <BODY>
```

In order to perform this combining, some scheme for the required separating has to be devised. For example,

```
HEADER (DATA) (DATA) (DATA).
```

In this example, each "(" points to its corresponding ")" by any scheme, for example, by the simple addition of a LENGTH field.

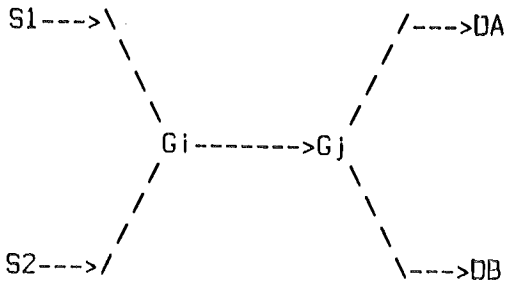
Obviously, the same combining could repeat indefinitely to accommodate an arbitrary number of small messages, up to the size limit imposed by the communication media.

At each level in the protocol hierarchy, there is an opportunity to provide a routing function. Most existing protocols do provide such a facility. However, most existing protocol processes also restrict their routing to be once per transmission unit (packet, message, segment, letter, etc.). To achieve efficient communication, we believe it will be necessary to provide for many level x packages to be communicated in one combined level x-1 package, and equip the level x protocol process with the means to separate the level x packages to various level x+1 processes.

It is also desirable for an intermediate process or gateway to combine similarly addressed packages. That is to say, packages from S1 to DA may be combined by GI with packages from S2 to DA. This must be done in a way allowing DA to discover the original packaging in order to be able to separate the packages.

A gateway should also be able to separate packages for different destinations which have the same initial route but distinct final routes.

In general, it is desirable for an intermediate processor or gateway to combine packages from different sources that have a common route, and for a subsequent intermediate processor or gateway to separate the packages when they no longer share a common route.



Here Gi may combine traffic from S1 or S2 to DA or DB in messages to be separated by GJ, which is the common subdestination for both DA and DB. The interesting thing is that S1 or S2 do not have to know about this commonality. It can be discovered only after the routing is performed, in this case by Gi.

This applies at any level, not only at the end address level as discussed.

Combining improves the efficiency of the transmission system in both the ends where the lower level of protocol processes handles fewer messages and in the middle where the network nodes or gateways handle fewer messages. The public packet networks charge for usage on a per message (or packet) basis for good reason. Combining messages directed to the same general destination results in lower charges.

A Grammar for Messages

In the previous sections, we presented some formal definitions for nesting and combining. These can be applied at any level for protocol nesting and combining. They can be merged to result in a very compact statement of nesting and combining. First we give a larger set of rules that make explicit the nesting and combining, then we give the rules resulting from eliminating redundant terms.

```
<MESSAGE> ::= <HEADER> <BODY>
<BODY>    ::= <DATA> | <COMBINE> | <NEST>
<NEST>    ::= <HEADER> <BODY>
<COMBINE> ::= <BODY> <BODY>
```

Eliminating redundant terms, we have the following grammar for a message:

```
<MESSAGE> ::= <HEADER> <BODY>
<BODY>    ::= <DATA> | <BODY> <BODY> | <MESSAGE>
```

If this grammar is used to generate and to parse combined and nested trees of messages, the address of each deliverable message is determined by all the headers between it and the root of the tree.

Suppose that both MESSAGE-A and MESSAGE-B are addressed to the same global destination, and possibly to different final destinations there (e.g., processes), then if both have identical HDR1 portions, these two MESSAGES,

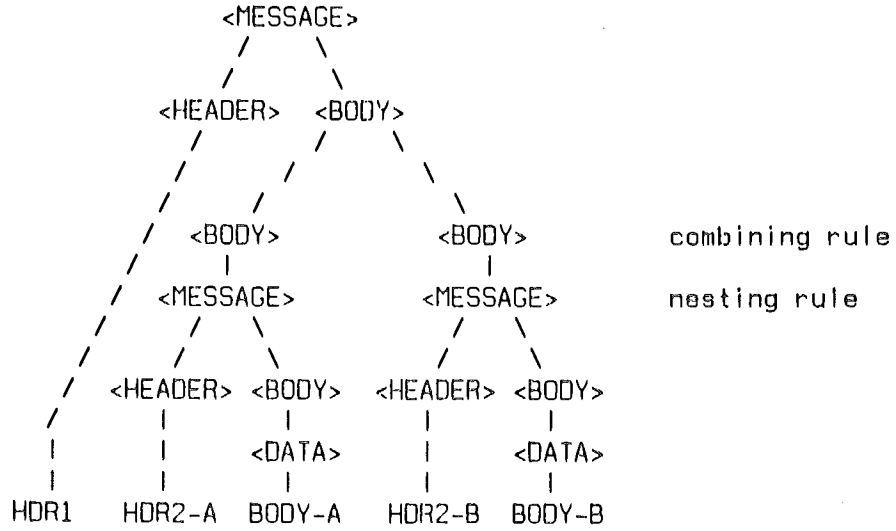
```
MESSAGE-A = HDR1, (HDR2-A, BODY-A) and
MESSAGE-B = HDR1, (HDR2-B, BODY-B)
```

can be combined into the single message:

```
MESSAGE = HDR1, (HDR2-A, BODY-A), (HDR2-B, BODY-B)
```

Thus we see that this combining is similar to factoring. The "common factor," the portion of the header that is identical for the two messages, is pulled out. This allows the saving of a few bits in transmission by having HDR1 only once and, more importantly, it reduces the number of messages to be communicated.

The tree representation of the generation of this message from the grammar illustrates both combining and nesting.



It is fairly easy to get confused about nesting and combining. Note that in the example, both MESSAGE-A and MESSAGE-B are already nested messages. From an application or user point of view, the BODY-A is important, and the rest is various header information. Yet from a structural point of view, the message has two parts, HDR1 and some body.

Therefore it is advisable to construct headers so that the more general (e.g., per connection) information is ahead of the more specific (e.g., per message) information.

Flow Control

Flow control is intended to protect limited resources shared between competing users. Usually, the resources of most concern are buffers and bandwidth. Flow control can take one of several strategies: (1) limit the

amount of new data, (2) limit the rate of data flow, or (3) limit the holding of old data.

Limiting the amount of new data transmitted involves a reservation or allocation mechanism. The reservation can be made a priori between the data sender and the data receiver, or the reservation can be made a posteriori according to the recent history of the communication. This form of flow control is used with applications like FTP and TELNET.

Limiting the rate of data transmission requires communication from the data receiver to the data sender of a specification of the acceptable rate. This form of flow control is found in applications like NVP.

Limiting the holding of data is usually implemented by discarding new messages (or denying their acceptance) until the older ones are taken by the destination process. Hence, in the case of FTP, the "tail" of the file is discarded until its "head" is successfully delivered. However, in the case of real time communication, such as sensor input, this policy is reversed, and the older information is discarded to make room for the newer, more up-to-date data.

Flow control has to be performed in each level separately. Only in a very limited set of applications, especially without any multiplexing, a flow control at a certain level can happen to protect adjacent levels too. This is the case because in the absence of multiplexing, the division between the different protocols layers is not necessary for the communication needs (such as addressing), even though it may be beneficial for other reasons (programming structure, etc.). Hence, since those levels could be merged, a single flow control mechanism may protect several of them at once.

However, this is obviously not the case when multiplexing is used, since flow control is essentially a per stream function, where streams are formed by communication between terminal addresses. If flow control is provided on the individual streams of communication at some high level, no protection is provided to a lower level, since many high-level streams may use that lower level.

Each level of protocol that provides a multiplexing function to several higher level protocols (in the sense of an address) also defines a new kind of package. Each level of protocol must provide for flow control of its kind of packages, on a per address basis, if reliable communication is to be achieved.

Flow control at level x is used to protect resources at level x and generally cannot protect the resources at any lower level. Therefore, the mechanisms used to implement flow control at any level must be unique to that level (that is, not shared with any other level).

The combining of packages of level x into a single package at level x-1 does not relieve the need for flow control to any level. It reduces the delays and improves the efficiency of communication.

Delays may be reduced because of fewer level x-1 packages being used to support the communication of level x. For example, if each package of level x-1 requires an acknowledgment before another can be sent, then significant delays accrue to subsequent level x packages while waiting for a level x-1 acknowledgment.

It is a dangerous mistake to implement a system that forces all processes whose messages are multiplexed together to behave according to the worst-case flow control. For example, suppose that at some level, messages of both the C1 and C2 communication paths are combined, and that at some higher level, the C1 path has to be slowed or even halted. Obviously, it is not desired to slow the C2 communication path also. It is even conceivable that this is the time to speed up C2. However, if some of the C2 communication messages are trapped with those of C1 in a way that cannot be undone or noticed, it might not be possible to separate these communication paths.

Therefore whenever combining is implemented, it is essential that the proper hooks should be left so that it is possible to "unbundle" it, to allow separation of communication paths when needed. This is, obviously, easier to achieve where duplication and/or loss of messages is allowed, since the messages blocked by a subset of their destinations would appear to be lost messages to the other destinations, which may invoke the appropriate action (e.g., retransmission) by the higher levels.

Summary

There are two aspects to multiplexing:

1. Nesting subdivides and expands the address space and provides common services to a multitude of higher level protocols.

2. Combining puts packages of data together in order to reduce the number of messages transmitted.

These two aspects of multiplexing may be implemented in any layer of protocol, either together or separately. Each of these aspects of multiplexing constrains the implementation options. Flow control should be used at each level where nesting is performed for multiple higher level protocols to protect the resources of that level and should be implemented with mechanisms that are independent of other levels. Combining messages reduces the number of lower level messages and so reduces cost and delay and improves transmission efficiency.

IMPACT

Protocol Analysis

We participated in an ARPA workshop on protocol verification and have identified some of the issues in protocol, as distinct from program, verification [31]. The issues involve timing considerations, process distribution, probabilistic behavior, and nontermination.

We also showed that it is very easy to specify overly restrictive protocols and that specifications of exactly what is intended may be quite difficult to construct.

Protocol Applications

The design for the prototype internet message system [4] is being considered by several other ARPA contractors and by the IFIP Working Group on Computer Messages Systems (WG 6.5) as a basis for their work.

We participated in an ARPA workshop on message systems, which concluded that the work on existing message systems should be minimized and that the focus should be shifted to the development of a new message environment like the one we proposed.

Protocol Design

The selection of the IP and TCP protocols by the DoD as the basis for a DoD internetwork protocol standard shows the impact of the work of the ARPA community on DoD communication systems.

Protocol Concepts

Through our participation in discussions at the Internet Working Group meetings and in technical meetings with other contractors, we have successfully influenced the development of conference-style communication using both streams and datagrams in mixed point-to-point and broadcast multinetwork systems. We have raised and expounded on the issues of multiplexing and source routing, which are now being considered by the Internet Working Group.

FUTURE WORK

Protocol Analysis

As a first step toward the verification of TCP we will study a series of example protocols of gradually increasing complexity. The protocol features found in TCP will be

introduced in this series of examples. We will attempt to use several protocol analysis and program verification tools on the example protocols. The tool we are considering for use include AFFIRM, GYPSY, and SPECIAL. These experiments in protocol analysis and verification will provide guidelines for the analysis and verification of TCP.

Protocol Applications

We will continue our experiments in internet message systems. We expect to develop a better understanding of the distinction between endpoint and relay message processing. We also will engage in experiments in multimedia messages to develop standards for representing and coordinating data of different media. To help in this development, we will cooperate in experiments in facsimile messages with UCL, BBN, and COMSAT, and experiments in voice messages with the NSC project at ISI (and possibly other NSC contractors). We will also consider the issues associated with internet file transfers.

Protocol Design

We will continue our contributions to the development of internet gateway level (IP) and host level (TCP, RTP) protocols. In particular, we will continue to support the Internet Working Group and the DoD protocol standardization efforts.

Protocol Concepts

We will collaborate with other projects on protocol issues. For example, we will continue our interaction with the NSC community on the development of an internet speech conferencing protocol. We will continue to raise important issues for discussion in the Internet Working Group and to seek out and develop new opportunities to use the capabilities of packet-switched communication systems.

REFERENCES

1. Cohen, D., and Postel, J., "Multiplexing Protocol," USC/Information Sciences Institute, IEN-90, May 1979.

2. Cohen, D., and J. Postel, "Source Routing," USC/Information Sciences Institute, IEN-95, May 1979.

3. Postel, J., "Transmission Control Protocol--Version 4," USC/Information Sciences Institute, IEN-81, February 1979. NTIS AD number A067072.

4. Postel, J., "Internet Message Protocol," USC/Information Sciences Institute, IEN-85, March 1979.

5. Postel, J., "Internet Message Protocol," USC/Information Sciences Institute, IEN-113, August 1979.

6. Postel, J., Message System Transition Plan. USC/Information Sciences Institute, JBP-64, February 1979.

7. Postel, J., Out-of-Net Host Addresses for Mail. USC/Information Sciences Institute, RFC 754, April 1979.

8. Postel, J., "Internet Datagram Protocol--Version 4," USC/Information Sciences Institute, IEN-80, February 1979. NTIS AD number A067849.

9. Postel, J., "Internet Protocol," USC/Information Sciences Institute, IEN-111, August 1979.

10. Postel, J., "Transmission Control Protocol," USC/Information Sciences Institute, IEN-112, August 1979.

11. Postel, J., "Internet Name Server," USC/Information Sciences Institute, IEN-61, October 1978.

12. Postel, J., "Multiplexing Protocol," USC/Information Sciences Institute, IEN-72, January 1979.

13. Postel, J., "User Datagram Protocol," USC/Information Sciences Institute, IEN-88, May 1979.

14. Postel, J., "Internet Name Server," USC/Information Sciences Institute, IEN-89, May 1979.

15. Postel, J., "Intenet Protocol Options," USC/Information Sciences Institute, IEN-114, August 1979.

16. Postel, J., "Address Mappings," USC/Information Sciences Institute, IEN-115, August 1979.

17. Postel, J., "Name Server," USC/Information Sciences Institute, IEN-116, August 1979.

18. Postel, J., TCP Meeting Notes--18 and 19 September 1978. USC/Information Sciences Institute, October 1978.

19. Postel, J., "Internet Meeting Notes--30 and 31 October 1978," USC/Information Sciences Institute, IEN-63, November 1978.

20. Postel, J., "TCP Meeting Notes--4 December 1978," USC/Information Sciences Institute, IEN-70, December 1978.

21. Postel, J., "Internet Meeting Notes--25 and 26 January 1979," USC/Information Sciences Institute, IEN-76, February 1979.

22. Postel, J., "TCP Meeting Notes--29 January 1979," USC/Information Sciences Institute, IEN-77, February 1979.

23. Postel, J., "Address Mappings," USC/Information Sciences Institute, IEN-91, May 1979.

24. Postel, J., "Protocol Options," USC/Information Sciences Institute, IEN-92, May 1979.

25. Postel, J., "Assigned Numbers," USC/Information Sciences Institute, IEN-93, May 1979.

26. Postel, J., "Internet Protocol Handbook--Table of Contents,". USC/Information Sciences Institute, IEN-94, May 1979.

27. Postel, J., et al., "TCP Implementation Status," USC/Information Sciences Institute, IEN-98, May 1979.

28. Postel, J., "Internet Meeting Notes--8, 9, 10, and 11 May 1979," USC/Information Sciences Institute, IEN-106, May 1979.

29. Postel, J., "Assigned Numbers,". USC/Information Sciences Institute, IEN-117, August 1979.

30. Postel, J., "Protocol Handbook Table of Contents," USC/Information Sciences Institute, IEN-118, August 1979.

31. Postel, J., Issues in Protocol Verification. ARPA Protocol Verification Workshop, March 1979.

32. Reed, D., and J. Postel, "User Datagram Protocol," USC/Information Sciences Institute, IEN-71, January 1979.

7. DISTRIBUTED SENSOR NETWORKS

Research Staff:
Danny Cohen
Jeffrey Barnett
Iris Kamony
Yechiam Yemini

Research Assistants:
Richard Brooks
Avishay Halavy
Daniel Schwabe

Support Staff:
Debe Hays

INTRODUCTION

Military and commercial organizations are looking forward to large integrated data processing systems in the 1980s. Such systems must bind together many separate applications so that information processing techniques such as correlation and synthesis are available across many domains. Ideally, the set of applications would be housed at a central location and users would gain access to this site through conventional network techniques.

Several factors and trends make the centralized systems approach less appealing, however.

1. Processor costs--a revolution in the cost/performance tradeoff for computer power is under way. The cost of cycles is rapidly decreasing and the cost/performance measure is optimum for small-to-medium-scale computers.

2. Communications costs--though the cost of communications should continue to decrease, the cost relative to cycles is increasing.

3. Locality--most data processing applications have the locality property. That is, each application accesses a relatively isolated portion of the data available to the whole system and each portion of the data is accessed frequently by only a few applications. Also, the accessed portion is very large compared to the input data and generated output.

4. Robustness--centralized systems are generally less resistant to component failure while decentralized systems can be more resilient.

For these reasons (and others), we believe that the large integrated systems of the future will be built upon a distributed hardware base and that control will be decentralized. Thus, these systems will use a paradigm of cooperating experts rather than one of masters and slaves.

Our investigation of these types of systems is focused upon Distributed Sensor Networks (DSN). A DSN naturally contains a geographically distributed set of sensors (not all necessarily of the same kind), communications mechanisms, and processes to combine information, perform correlations, and draw inferences. A DSN may also include command and control components as well as providing for the fusion of DSN information with information gathered outside the system proper, i.e., intelligence. Thus, research into DSN provides a rich source of problems, the solutions to which enhance our ability to design and build the large integrated systems of the future.

Overviews of the problems and issues inherent in DSN systems are found in [2] and [16].

PROBLEMS BEING SOLVED

Even though we believe the large integrated systems of the future will be distributed and decentralized, our knowledge is not yet sufficient to design THE system. Therefore, our efforts are directed toward the more general problems: in particular, system issues that are both sensor- and scenario-independent. Sensor independence is possible because diverse sensor families behave similarly when their behavior is abstracted [7]. Scenario independence is dictated by our desire to examine large design and problem spaces encompassing many applications.

Our research is concentrated on four general system design and methodological areas and two specialized studies. These areas and studies are described below.

The Communication Problem

A DSN is an integrated system in that the total system, including the communications mechanism, works toward a common objective. In an ordinary system, the communication mechanism provides services to a variety of unrelated applications competing selfishly for resources. Further, a DSN is a realtime system where the applications often specialize in determining the relative importance of data. Thus, it is quite clear that application knowledge must affect the behavior of the communication services in areas such as flow control and priorities. The DSN communication problem is how to provide for the interaction of application knowledge and communication for the good of the entire system.

The Architecture Problem

The implementation of a distributed and decentralized system must include a decomposition methodology. Each component in the decomposition is described by the role it plays in the total system. The description includes the input requirements for the component and the output it produces. The architecture problem is to determine reasonable guidelines for decomposing a system given the system's global objective, the available resources, and the types of algorithms to be employed.

The Organization Problem

Even given workable solutions to the communication and architecture problems, many system issues remain. The deployment and reallocation problems, together called the organization problem, are chief among them. The deployment problem is how to assign components to the available resources. For example, how many instances of a particular class of process should there be, at which node should each run, and which other process instances are their communications partners? The reallocation problem is similar. Namely, how is the system redeployed in the face of component failure, the introduction of new components, or a specialized tactical need? A DSN faces this problem because the critical nature of its objective may not allow the system to go offline to reallocate its resources. A system busy monitoring and tracking craft over a large geographic area should not lose its entire working state because a few components fail or are compromised.

The User Interface Problem

Decentralized systems present a problem to their users. The data presented to the user is often gathered from many diverse sources. It is correlated and perhaps inferences have been drawn from the data on its way to presentation. The user must be able to determine the sources, processes applied, and the general level of confidence that should be attached to the summary he sees. Also, the user must be able to advise the system about its future behavior, for example to focus attention on a particular area. Further, the user can be the source of intelligence not normally available to the system, e.g., introduction of a new flight plan. Thus, the user of a DSN must be able to understand the system-generated information as well as to control the system toward fulfillment of the overall goal.

The Position-Location Problem

Communicating packet radios can measure the distance separating them with a high degree of accuracy--an absolute error of less than twenty feet. This distance measurement capability can be put to good use. One use is computing the exact location of equipment, particularly sensors, after the equipment is deployed. Contrast this to installing the equipment exactly at predetermined locations--not always possible in rugged or hostile terrain. Another usage is determining the present location of craft, particularly those engaged in surveying missions and thus needing an accurate measurement base.

The abstract position-location problem is: given a set of nodes, the distance measurements between some of the nodes, and the locations of some of the nodes, (1) determine the set of nodes whose absolute position can be computed, (2) compute the location of all such points, and (3) determine the stability of the solution.

The Distributed Algorithm Problem

The paradigm for a distributed and decentralized system is to take a process we presumably know how to implement in a centralized fashion, break it into pieces, distribute the pieces, and produce an effect similar to the original. The distributed algorithm problem can be stated as: given a class of system organizations and restrictions on the communication facility (e.g., number of messages), when is it possible and desirable to distribute particular parts of the process. Today, we have no substantial theory about what processes can and cannot be distributed subject to constraints.

GOALS AND APPROACH

The major goal of the DSN project is to develop and export technology aiding the implementors of distributed and decentralized systems. The first step in this direction is derivation of an adequate description of the DSN problem space. This includes not only detailed knowledge of what phenomena a DSN must confront and what types of components are available, but also understanding why some systems are more highly valued than others. In other words, objective methods of evaluating a system's performance must be developed.

Three techniques are available to gain understanding of complex systems. In order of increasing desirability, they are

1. Simulation--describe and make operational approximations to system components.

2. Testbed--provide an environment for testing some components while simulating the behavior of other components.

3. Analysis--mathematical description and solution.

Simulation of an aircraft is done by writing the sets of partial differential equations (obviously approximations) describing the behavior of the components, their interaction with the environment, and then numerically integrating the equations to observe the time history of stress parameters. A testbed for an aircraft is a wind tunnel. Actual components, e.g., a wing, are tested in a realistic environment. A testbed is component-independent because it can be used to test a variety of components--a desirable property. For example, a wind tunnel is usable for different wings as well as rudders and nose cones. Thus, the value of a testbed transcends its use for testing a single item.

Closed-form analysis for a system as complicated as an aircraft is rarely possible. Unfortunately, such analysis of DSN is not possible for the same reason--complexity. However, a few subproblems in the DSN world are amenable to closed-form analysis (for example, our work on position location and development of a theory of distributed computation). Both problems are characterized by precise mathematical definition, without which analysis is impossible. We are working on a set of algorithms for use by implementors of packet radios. These algorithms, called welders, use the distance estimating capabilities of packet radios to determine the topology of a network. The work on distributed algorithms has proceeded to the point where the problem can be stated in reasonably precise terms. Our goal is to develop the theory deeply enough so that results are applicable to real-world-sized processes.

Unfortunately, the research into system issues--architecture, communications, organization, and the user interface--cannot employ such techniques. Therefore, we propose to use a combination of simulation and testbed. Objects and components in the DSN environment such as targets, sensors, and communications devices (e.g., packet radio hardware) are always simulated. System components that are software processes

can either be simulated or the actual program can be run. The testbed is our major design analysis tool. Provisions are made for several important activities:

Configuration	Allow system design to be specified from a data base of component representations.
Dynamics	The configuration must be able to dynamically change.
Communications	Allow simulation of a variety of communications regimes.
Resources	Allow and properly account for the fact that many processes and activities compete for resources at a node (a single computer).
Debugging	Testbed components must be debugged. This includes instrumentation and testing tools for the component developers.
Reporting	Summaries of resource utilization, system performance, etc., must be available.
Interface	It must be possible to construct and use realistic user interface(s), includes online debugging and probing by the simulated system user.

The testbed is basically an event-driven system with special provisions for handling the problems of local resource contention occurring at the system nodes. Global contention appears in the use of the available communication bandwidth and is handled by the communications modules. Since the testbed is not committed to a single communication paradigm (e.g., TCP or HEARSAY II), the testbed user must supply these modules.

The testbed provides a set of interfaces to its users. They allow a "patch panel" style of system configuration to be used by isolating components from specific implementation decisions. Three interfaces are provided.

The first is the environment interface as shown in Figure 7.1. It is used by the representations of communications and sensor processes to access target and geographic information. The shaded areas are a fixed part of the DSN testbed and are invariant for different design configurations. The unshaded areas are to be defined by the DSN designer to emulate a particular system configuration capability (i.e., explicit sensors and a specifically defined communication medium) that will operate in a given geographic region in which specific targets will appear.

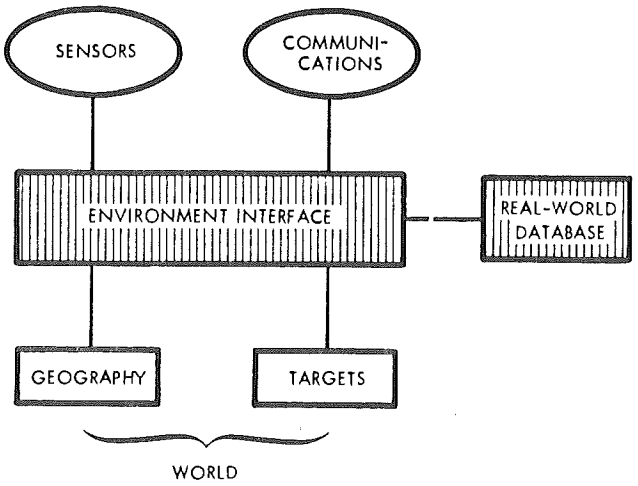


Figure 7.1. Testbed environment interface

A report and debugging interface (see Fig. 7.2) is used to collect and deliver information within the system to its developers and its designer/user. Modern system implementation practice dictates that processes are built with debugging and data probe hooks. This interface is our institutionalization of this practice. The interface also supports full data collection and performance reports about its capabilities.

Figure 7.3 is a simplified view of the modules that handle resource allocations in a DSN system. A communications component transports messages between application processes, each of which resides on a node and is controlled by the operating system on the node.

The testbed kernel, shown in Figure 7.4, is the third interface; it controls the scheduling of application processes and the communications between them. Nodes that are independent processes in the real world are scheduled as parallel co-routines by the testbed kernel. They are activated at the same simulation time. Messages are sent from applications and delivered to nodes at the correct simulation time. The parts of the testbed kernel supplied by the designer/user are the node operating system and the communications module. Communication, as shown in Figure 7.4, is separated into three parts: the transportation mechanism, the node support communication module, and the application level communication interface.

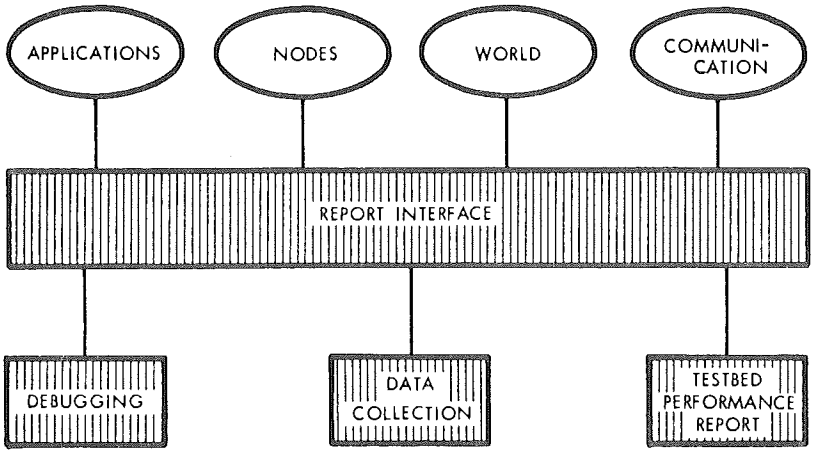


Figure 7.2. Testbed report interface

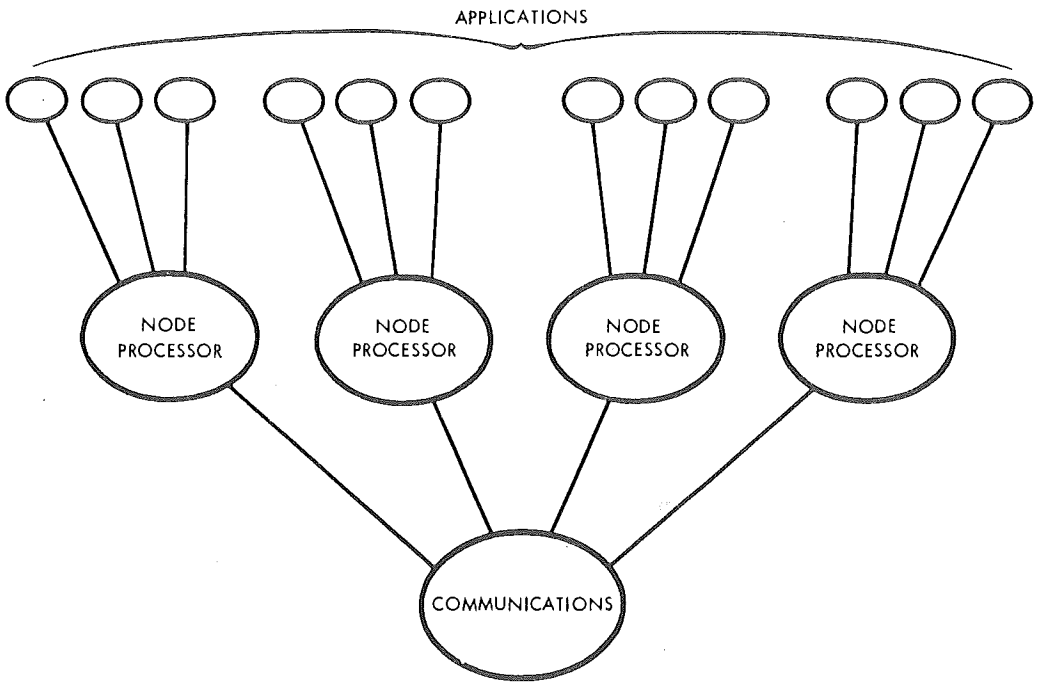


Figure 7.3. Basic view of testbed

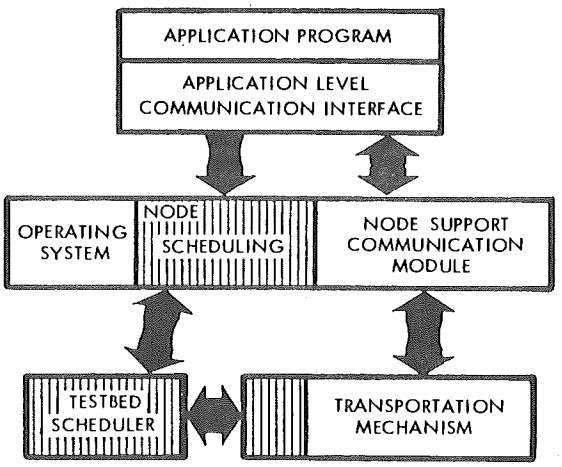


Figure 7.4. Testbed kernel

A request to communications (via the application level communication interface) by an application program has two parts: (1) a data request, say to read a buffer, and (2) a control part, (e.g., to put the process to sleep until a particular event such as a message arrival or a time-out occurs). The node support communication module is given the control part of the request and insures that the associated activities occur at the proper simulated time. The data request part is handled by the transportation mechanism, i.e., when a message arrives it is delivered to the node at the correct simulation time.

An important problem for the designers of DSN is how application knowledge is to interact with resource allocation. For example, what functionality does the communication system provide to the user programs? Here, too, interface conventions are important. A goal of the project is the investigation of such issues. The constraints are that the interface (language) must be rich enough so the application can describe its knowledge to the underlying mechanism, and it must be simple enough so that application programmers (not necessarily network specialists) can use it.

Figure 7.5 shows the total DSN testbed design. On the right are the system tables. Definitions of the different node, application, and communication modules are kept in a

database. Part of the DSN testbed is an interactive building component that communicates with the DSN designer to allow him to describe a particular DSN configuration. The DSN builder also interacts with the designer to define an environment (geography and targets). When the system configuration and environment are defined and entered into the corresponding system tables, then the DSN testbed system is "configured" and the simulation session begins.

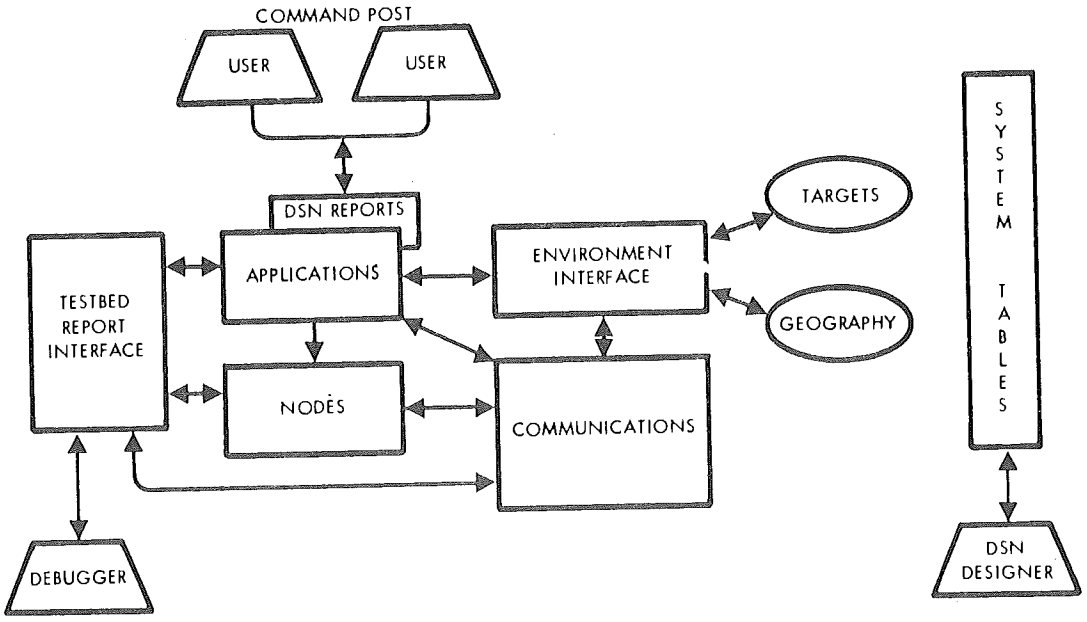


Figure 7.5. DSN testbed design

The testbed kernel (as described earlier) is composed of the nodes and their applications and the communication components. As shown in Figure 5, an application may reside on a command post node to support users who can get reports of DSN activity and also send requests and commands to the system. An important part of the testbed system design is to determine the necessary user functions and to develop a human engineering interface that allows for graceful interaction. We currently plan to make use of the ISI situation display capability for graphic presentations to the user at the command post level.

Figure 5 also shows how the environment interface and the testbed report interface fit into the total picture. Not shown in the figure, but planned, is a database in which the report interface will enter pertinent information about each simulation run.

With the testbed, we can investigate many system-level problems. Competing architectures, communications strategies, organizational paradigms, and user interfaces can be compared. From this, we can state and quantify some of the design tradeoffs for DSN and other decentralized systems.

PROGRESS

The FY79 activities and accomplishments of the DSN project are described below. A collection of working papers written by project members is available in [8]. Some of the papers appear also in the open literature or have been presented at conferences.

Progress on Defining the Problem

The initial task of a project such as DSN is to define the problem to be solved and the scope of interest. In [2], the problem is formulated as one of turning errorful observations derived from many sources (e.g., sensors and intelligence) into a reasonable world picture. The problem space is defined to include not only sensors and low level processes, but also decision-making functions. Several advantages of distributed and decentralized systems are discussed in terms of system scenarios that take advantage of the unique capabilities.

In [16], technological problems particularly relevant to the DSN world are enumerated. These are the areas where progress must be made before DSN are a reality. The core problem uncovered is distribution of inference procedures.

In [18] and its companion paper [3], the problem of an objective function for evaluating DSN systems is addressed. The major point is that the value of a system depends upon its intended usage. Though more precise position estimation is obviously a benefit and additional cost is an obvious liability, no more can be said until the value of precision is known, and that value depends upon the use of the extra bits. Also discussed is a methodology for extracting information from the ultimate users of the systems to construct an objective evaluation criterion.

Another aspect of defining the problem is determining what phenomena can be ignored. In [7], many different sensor families are reviewed. It concludes with the assertion that nearly all sensors have a similar enough functionality so that differences can be safely

ignored. In particular, sensors are well described by their sensitivity and error distribution. This is indeed fortunate because it allows us to investigate the high-level system issues without committing large amounts of resources to detailed sensor simulation.

Progress in System Methodology

The architecture problem for distributed systems has been a core computer science topic for many years. See, for example, [1], [12] and [13]. However, the more complex issues of decentralized control and decisionmaking have received less attention. Though these problems are central to DSN, we have made little progress because of the lack of adequate tools. This lack will be overcome by the availability of the testbed in the near future. In the meantime, several simple simulation exercises have been attempted with an ad hoc system. The main result is our recognition that a good experimental framework providing excellent control and monitoring facilities is necessary.

The system organization problems of deployment and reallocation are investigated in [4]. The problem is formulated in terms of capabilities, where a capability is a description of a simple resource, e.g., processor power, memory, or bandwidth. A system is viewed as a collection of players that offer capabilities (a computer node is an example of a player that offers capabilities) and a collection of roles that consume capabilities. A role is a functionality such as can be provided by a process. A role is described not only by the capabilities it consumes but by its input/output relations to other types of processes. Thus, the deployment problem is stated as finding an assignment of roles to players that best satisfies the system's objective. The objective describes aspects of necessary system behavior and tradeoffs among competing alternatives.

Progress on the Testbed

The testbed is the major tool necessary to achieve the long term goals of the project. The top-level design is completed (see [5]) and implementation has begun. The core of the testbed is the mechanism it provides for handling time. Two problems make time maintenance complicated: (1) resources (capabilities) at a node are shared, therefore the placing of events in time is dependent upon a resource sharing policy and (2) simulation of communication may be at a different grain size (minimum significant time) than the rest of the system.

A detailed design of the time handler is complete--in a semiformal specification language--and implementation in SIMULA has begun. The implementation provides for modular representations of a node's resource sharing policy as well as an interface for simulating a variety of communication regimes.

Initial attempts have been made to design a configuration language to describe system topology using the capability model described in [4].

Progress on Communication Methodology

A large effort has been expended by the project on communication mechanism design because a DSN presents two unusual features: (1) the entire system, including its communication components, has a common objective and (2) the system is realtime. Several investigations into these and related problems are described below. More details and information are available to the interested reader in the documents cited.

In [15] the topic of high-level protocols is investigated. Three subproblems are addressed: language, coding, and transportation. Some examples of current practices are given, and it is argued that modern techniques for expressing structure and control in programming languages can and should be applied to analogous problems in communications among application processes in a network. The motivation for the conclusion is the need for resource sharing and common objectives among distributed processes.

In order to provide an appropriate interface to application level programs in DSN or other distributed systems, a communication interface language must be provided between the applications and the underlying communication mechanism. A very important aspect of this language is naming, since processes typically do not have static knowledge of which other processes can supply needed input or employ their output to advantage. In [14] a naming scheme is proposed based upon application-determined labelling. A process identifies itself to the communication system by specifying a set of labels describing its interests, e.g., position-estimator. The labels are used to search for communication partners sharing common interests and to simplify dynamic reorganization.

In [9], protocol specification issues are discussed. It is shown that protocol designs respond to our knowledge of real-world behavior of networks, e.g., lost or duplicated

messages, out-of-order deliveries, and random delays. This presents a difficult problem to the designers of protocols and networks. How are formal specifications and correctness criterion to be expressed? This theme is continued in [19] where the problem of formal mechanisms to reason about protocols, particularly their time-dependent behavior, is discussed.

An important theoretical result is derived in [20]. The "Generals Problem," formulated by R. Gallager, is defined and discussed. Given two divisions of the same army on either side of an enemy division, how are the two generals going to coordinate their attack? The only means available is to send messengers back and forth through enemy lines, hence the probability of message receipt is less than unity. It is shown that there is no sequence of message passing such that both armies are certain of the time for a synchronized attack. In other words, there is no possible protocol for guaranteed synchronization of distributed processes with an imperfect communication mechanism.

In [10], the problem of flow control in distributed realtime systems is investigated. It is concluded that there exist problems that cannot be solved without application-level knowledge. In particular, there are (at least) two different kinds of messages: those whose value decrease with age and those whose value does not. An example of the first kind is a position estimate that has been succeeded by a later estimate. The old message can be discarded. On the other hand, early messages should be retained in normal (sequential) applications such as file transfer. If later messages are discarded because of congestion, they can be retransmitted, while discarding earlier messages can cause congestion at message assembly sites.

The problem of decentralized access control schemes for shared channels is discussed in [21]. The basic issue is, When should packet radios transmit and when should they delay? If transmissions of proximate radios overlap in time, a collision occurs and both messages can be lost. On the other hand, unnecessary delays sacrifice a portion of the available bandwidth. An optimal algorithm is developed for making access decisions employing local knowledge of the network state and the probable priorities of messages.

Progress in Position Location

Our point of departure is that the task of position location would be implemented in Packet Radio Networks which would not wish to commit to it more than a minimal amount

of its processing and communication resources. At the extreme case, this assumption implies that distance measurements and position-location computations will be carried out infrequently, which renders the tracking approach unattractive (you would not wish to use outdated position estimates to filter your present position). Therefore we have preferred to study a primitive, less favorable, form of the problem where the computation of positions is based upon present distance measurements only.

We have derived mathematical solutions to the following problems. Given a set of nodes and a set of distance measurements between them, determine which groups of nodes can be positioned (i.e., their relative location computed); in what order can the relative positions of nodes in a group be computed; how can the positions of positionable groups be computed. In addition, we have obtained a rich set of theoretical results of major significance to problems of position location [Yemini 79e-h]. Some of the major novelties introduced are the use of efficient probabilistic algorithms that compute answers to difficult (probably intractable) problems, incurring potential error but with very small probability; that is, we trade accuracy (with small probability we may not get a correct answer) for speed.

We have turned the theoretical results into practical algorithms and implemented those in SIMULA. The present version of our position-location system runs a centralized form of the position-location algorithms. It does not yet reflect the full power of the theoretical results derived; nevertheless, it already surpasses in capabilities any other position-location algorithms in the literature. The ultimate system could compute the position of any theoretically solvable position-location problem.

Progress in Distributed Computation Theory

A class of evaluation (estimation) algorithms called E-functions are axiomatically defined in [6]. E-functions include the general class of statistical descriptions of populations such as means and the median. A theory of E-functions is developed and related to classic work on inequalities. A result of particular significance to distributed systems is the statement of necessary and sufficient conditions such that the computation of an E-function can be distributed. It is shown also that there exist E-functions that cannot be distributed in any reasonable way.

Our current work in the theory of distributed computation concentrates on two mathematical models of distribution. The first views the communication scheme as a restriction on a functional composition and allows the theory of functional equations to be applied. The second model views the problem as one of the topology of equivalence classes. The second method employs results from the theory of transformations.

IMPACT

The DSN project has a long-term commitment to the problems of distributed and decentralized systems. As our goals are met, the ability to design and analyze these systems will move from the realm of research to practical engineering development. Our main contribution will be to furnish some of the methodology to make this transition possible. In particular, we are working on the general system design problems as well as techniques to objectively evaluate alternative systems.

In the short run, we are developing several tools that can be used by others. The position location algorithms for use in packet radio networks are near completion. Also, the testbed will be available for use by other investigators via the ARPA network.

FUTURE WORK

As with any project with long term goals and commitments, our future plans are a continuation of present work. The major discontinuity is the completion of the testbed so that problems of decentralized system methodology can be actively pursued. Initial experience with the testbed will lead to improvements in the testbed itself.

Two primary goals for the near future are to (1) provide a user interface that aids a variety of experimentation with the testbed and (2) implement a high-level protocol language to buffer application processes from expected frequent changes to the communication simulation. The design of the user interface will build on the situational graphics done at ISI and the interaction mechanism provided to the user will be based upon conversations with current users of sensor based systems.

The goal of position-location research in the near future is to bring the study of the centralized algorithms to conclusions and then use the software testbed to study the position-location problem in the more realistic setting of decentralized environment. In addition we plan to apply our results to the problem of position tracking, to overcome the

shortcomings of present systems. The algorithms and theoretical studies will be documented and made available for use in a variety of applications.

Our work on the theory of distributed computation will continue with an attempt to model realistically sized processes and determine conditions where decentralization is possible and profitable. A similar problem is addressed using the testbed but in a more pragmatic way. Alternative system architectures and organizations will be compared so that tradeoffs can be established. Our major task in the future is identifying and analyzing these tradeoffs in a way that allows others to design and deploy decentralized systems.

REFERENCES

1. Barnett, J., "Module linkage and communication in large systems," in Reddy, R. (ed.), *Speech Recognition*, pp. 500-520, Academic Press, New York, 1975.

2. Barnett, J., "DSN problems--an overview," in *Distributed Sensor Networks*, pp. 37-40, Carnegie-Mellon University, December 1978.

3. Barnett, J., "The objective consumer," in *DSN-Distributed Sensor Networks: Working Papers*, USC/Information Sciences Institute, WP-12, 1979.

4. Barnett, J., "DSN: The organization problem," in *DSN-Distributed Sensor Networks: Working Papers*, USC/Information Sciences Institute, WP-12, 1979.

5. Barnett, J., "An environment for designing and evaluating DSNs," in *DSN-Distributed Sensor Networks: Working Papers*, USC/Information Sciences Institute, WP-12, 1979.

6. Barnett, J., "Evaluation functions," in *DSN-Distributed Sensor Networks: Working Papers*, USC/Information Sciences Institute, WP-12,, 1979.

7. Cohen, D., "On various sensors," in *DSN-Distributed Sensor Networks: Working Papers*, USC/Information Sciences Institute, WP-12, 1979.

8. Cohen, D., J. Barnett, Y. Yemini, and D. Schwabe, *DSN-Distributed Sensor Networks: Working Papers*, USC/Information Sciences Institute, WP-12, April 1979.

9. Cohen, D., "Protocols for dating coordination," in *DSN-Distributed Sensor Networks: Working Papers*, USC/Information Sciences Institute, WP-12, 1979.

10. Cohen, D., "Flow control for real-time communication," in *DSN-Distributed Sensor Networks: Working Papers*, USC/Information Sciences Institute, WP-12, 1979.

11. Cohen, D., "An algorithm for determining the rigidity of 2-dimensional structures and the stiffness of 2-dimensional graphs," in *DSN-Distributed Sensor Networks: Working Papers*, USC/Information Sciences Institute, WP-12, 1979.

12. Estrin, G., "A methodology for the design of digital systems-supported by SARA at the age of one," in *Proceedings of the National Computer Conference*, AFIPS, 1978.

13. Lesser, V., *PCL: A Process Control Language*, University of Massachusetts at Amherst, Technical Report, 1979.

14. Schwabe, D., "Communications interface specification," in *DSN-Distributed Sensor Networks: Working Papers*, USC/Information Sciences Institute, WP-12, 1979.

15. Sproull, R., and D. Cohen, "High-level protocols," *Proceedings of the IEEE* 66 (11), November 1978, 1371-1386. Special Issue on Packet Communication Networks.

16. Yemini, Y., "Distributed sensor networks (DSN): An attempt to define the issues," in *Distributed Sensor Networks*, pp. 53-60, Carnegie-Mellon University, December 1978.

17. Yemini, Y., "The positioning problem-a draft of an intermediate summary," in *Distributed Sensor Networks*, pp. 137-145, Carnegie-Mellon University, December 1978.

18. Yemini, Y., "A study of the DSN objective problem," in *DSN-Distributed Sensor Networks: Working Papers*, USC/Information Sciences Institute, WP-12, 1979.

19. Yemini, Y., "Issues in protocol design for DSN--a preliminary study," in *DSN-Distributed Sensor Networks: Working Papers*, USC/Information Sciences Institute, WP-12, 1979.

20. Yemini, Y., and D. Cohen, "On some issues in communication between distributed processes," in *Proceedings of the First International Conference on Distributed Processing*, IEEE, 1979.

21. Yemini, Y., and L. Kleinrock, "On a general rule for access control, or silence is golden," in *DSN-Distributed Sensor Networks: Working Papers*, USC/Information Sciences Institute, WP-12, 1979.

22. Yemini, Y., and D. Cohen, "On some algorithmic aspects of structural rigidity," in *DSN-Distributed Sensor Networks: Working Papers*, USC/Information Sciences Institute, WP-12, 1979.

23. Yemini, Y., "On some combinatorial aspects of structural rigidity," in *DSN-Distributed Sensor Networks: Working Papers*, USC/Information Sciences Institute, WP-12, 1979.

24. Yemini, Y., "On some randomly decidable geometrical problems," in *DSN-Distributed Sensor Networks: Working Papers*, USC/Information Sciences Institute, WP-12, 1979.

25. Yemini, Y., "Some theoretical aspects of position-location problems," in *Proceedings of the IEEE 20th Foundations of Computer Science*, 1979.

8. AUTOPSY

> In spite of the great wealth of our languages, a thoughtful mind is often at a loss for an expression that should square exactly with its concept; and for want of which he cannot make himself altogether intelligible, either to others or to himself.
>
> *Critique of Pure Reason*, Immanuel Kant

Research Staff:
Stephen D. Crocker
Richard Gillmann
Jeffrey V. Cook
Craig Taylor

Consultants:
David F. Martin
Hanan Samet

Support Staff:
Joyce K. Reynolds
Barbara Rush

OVERVIEW

Problem Being Solved

The goal of the Autopsy project is to develop tools to aid in translation of programs from one language to another. To provide a concrete basis for the project, attention is focused on translation of CMS-2M programs into Ada. Eventually, we envision generating translation tools automatically from formal definitions of the old and new languages.

We seek to answer the following questions along the way.

1. How should a translation system be designed? What interactions between the user and the system are expected? Which tasks can be performed by the system and which tasks must be carried out by the user? These are the *system questions.*

2. What are the theoretical issues in translation? To what extent can the definitions of the old and new languages be formalized? Can formal definitions of the old and new languages be used to generate a translation system? These are the *theory questions.*

3. What programs need to be translated? Who will carry out the task? What level of effort is acceptable? How will translation fit into the development and maintenance cycle? These are the *usage questions.*

System Design

We believe there are two constraints guiding the design of a useful translation system:

1. A translation system must do *almost all* of the work.

2. No translation system can do *all* of the work.

If the translation system does not do most of the work, it will not provide sufficient leverage to the user. The purpose of translating a program instead of rewriting it manually is to save time and manpower. Since there are often reasons to favor rewriting a system, the translation must be quite competitive with hand rewriting to be useful at all.

On the other hand, there are many hard problems in translation, some of which are not even potentially solvable. For example, some code contains timing loops whose correctness depends on the speed of the machine and the exact code emitted by the CMS-2M compiler. Detecting the intent of the code is impossible, since the functional behavior of the code is legal but irrelevant.

As another example, some CMS-2M programs are really sets of processes connected through calls to the operating system. Unless the semantics of the operating system and the run-time binding of the processes are specified, the critical information is not even present. However, even if this information were present, the transformations required to accomplish the same tasks in Ada are well beyond any forseeable abilities of a purely automatic translation system.

To weave a path between these two considerations, we envision a translation system that is semi-interactive in its basic structure, with powerful tools provided to carry out as much of the work as can be automated. The user in charge of the translation enters his programs and then directs the application of the tools. Our view of the tools includes the following possibilities:

Direct Translator — This is the tool that carries out automatic translation on the bulk of the old program. When it encounters untranslatable constructs, it must encapsulate them so they may be treated later by either the user or other tools. The direct translator should also be designed

to work on portions of programs so it may be called by the user after preliminary or intermediate modification of the old program has been carried out by the user.

Editor — At the other end of the spectrum, the user must have recourse to make any modification to either the old code, the new code, or any intermediate representation. Complete recording of all editing commands is necessary to provide a history of how the new program was derived from the old program.

Verifier — Verification technology is mature enough to provide some limited but useful tools. If the user substitutes new code for old by hand, but does so in a fairly mechanical way that preserves the basic algorithm, it is possible to mechanically verify the equivalence of the two programs. Samet's work [18] provides the basis for this tool.

Pattern-Directed Transformer — The user would direct the application of certain transforms to all places that meet the criteria for application of the transforms. If the user may write his own transforms, this tool is essentially a programmable editor. If the set of transforms is restricted to an "approved" set, this tool becomes a user-directed translation system.

These tools must exist within a coherent framework. We envision the framework to be a monitor that records all of the transactions and provides an "undo" capability. When any session is complete, the monitor records the list of transactions as part of the information stored about the program being translated. This "audit trail" is replayable and forms an accurate basis for understanding the exact relationship between the old and new programs.

Details of the status of the direct translator are given in the next section.

Formal Definitions

Formal definitions of programming languages play several roles in our work.

1. The exercise of preparing formal definitions of the old and new languages forces into view a number of details that might otherwise be misunderstood. Errors in our translation tools are thereby avoided.

2. The structure of a formal definition may give some guidance for the structure of the translation system.

3. Only limited experience with formal definitions exists. Attempts to build formal definitions point up difficulties with present formalisms. It is envisioned that formal definitions will eventually be machine processable to derive translation and verification systems.

Our work with formal definitions has involved development of both formal definitions and tools to process formal definitions. We have also watched related work quite closely--particularly the development of the Ada formal semantic definitions by the IRIA group--and imported tools developed elsewhere. Further details are given in the last section of this report.

Progress

- An automatic translation tool has been developed.

- An intermediate language form representing both CMS-2M and Ada programs has been designed.

- A specialized language for the translator, APPSS, has been created. We comment on this below.

- We have spent quite a bit of time testing and debugging pre-existing CMS-2M tools and removing nits from completed CMS-2M code.

- No engineering of an interactive, multi-tool system has yet taken place.

- Design of the verification tool has begun.

- We have begun a serious translation experiment ourselves.

- We have made quite substantial progress in developing theory and tools for writing and using formal semantic descriptions.

- We have begun to exercise these tools in the preparation of a full-scale formal semantic definition of CMS-2M. This effort has helped in the understanding of CMS-2M and thus with the translator.

Reflections

In addition to the specific products of the effort, we also have some perceptions to report.

1. At the outset, we envisioned an interactive system in which the user would combine a variety of tools to handle various problems in a translation. The user would, of course, have to be knowledgeable about the utility of each tool, but he would also have to be able to examine the code easily and understand what issues face him.

   We have seen that CMS-2M programs are generally quite large. Interactive pawing over them is not likely to be easy. Accordingly, we are not quite so optimistic about the contribution of just the system framwork; the tools themselves had better be very powerful.

   We have not abandoned our view that the system should be interactive, that the steps in the translation process should be recorded, able to be undone and replayed, and that the user should have control to direct the process. However, a system with just these characteristics but without powerful tools to carry most of the burden will not be useful.

2. We had originally viewed the intermediate language we've developed as serving just our own needs and not having any significance outside of our project. Recent discussions with others now suggest that our development may be more useful than we had thought.

   In the future, it is likely that tools dealing with Ada code will need to communicate intermediate representations instead of source code. There is some pressure to choose satisfactory intermediate representations as early as possible. To do so, it is necessary to examine the intermediate representations now in use in diverse applications to see what's essential and what's idiosyncratic.

   Other representations to be considered are TCOL $_{Ada}$ in use at CMU [20] and the IRIA abstract syntax [6].

3. The development of the APPSS language for the translator may also have significance beyond its initial purpose. In considering the possibility of generating a translator automatically from the formal semantic descriptions of the old and new languages, we can ask what language should be used for the generated translator. We had not given this question much thought at the outset of the project, but it now appears that APPSS might be the language of choice.

   The translator generator is likely to a heuristic program that analyzes the formal semantic definition of the old language and searches for corresponding components in the new language. It seems reasonable to organize the results of each search in terms of one or more productions. The entire set might then carry out translation. There is much work to do, of course, but we find it encouraging that a production oriented language was invented as a convenience for writing the first translator by hand.

4. Part of the reason for selecting CMS-2M as the old language was the perceived availability of both CMS-2M code and of CMS-2M tools on the ARPANET for compilation and execution. CMS-2M code has been somewhat hard to find, as it appears to have been in use for a relatively short period of time. At the same time, the tools on the ARPANET have never been exercised thoroughly. All of the usual problems in smoothing the interfaces and discovering the necessary but undocumented details have been encountered. Although this effort is costing us far more than we had ever dreamed, the positive effect will be that the set of tools will be in better shape.

Future Work

During the coming year, an interactive monitor will be designed, implemented, and attached to the direct translator and one or more existing editors. A simple equivalence verification system will be developed and tested.

A larger set of CMS-2 programs will be translated. However, comparison of execution results between old and new code will not be possible until Ada compilers are ready for use.

During this time we will also investigate the extension of the system to CMS-2Y and to languages of interest to the Air Force, e.g., J73.

We will continue our development of the formal definition of CMS-2M, and we will continue to study the IRIA formal definition of Ada. We expect to have an interpreter for the IRIA formal definition in order to facilitate testing and experimentation by compiler implementors and others needing to understand the precise semantics of Ada.

THE DIRECT TRANSLATOR

System Overview

The Autopsy Direct Translator (ADT) is a computer program that automatically translates CMS-2M programs into Ada. When it is unable to translate certain parts of the source code, it leaves a descriptive comment at that point in the output. The user is then expected to complete the translation by hand.

The ADT translates a program in four steps, involving five different representations. The representations are

1. CMS-2M source code
2. CMS-2M parse tree
3. Autopsy IL
4. Ada parse tree
5. Ada source code

The first of the four passes is the parser, which produces a concrete syntax tree from CMS-2M source card images. The second pass translates the concrete syntax into Autopsy *Intermediate Language* (IL), a tree structured representation of the program. The third pass translates IL into an Ada concrete syntax tree. The fourth pass prettyprints the Ada tree into a formatted Ada program.

The interesting parts of the translation are passes two and three. These tree transformations are written in a special language developed for Autopsy, called APPSS (A Pattern-directed Production System for S-expressions). APPSS is described below.

The system is structured in a way allowing its extension to include languages similar to CMS-2M. For example, JOVIAL, a likely candidate, could be easily translated into Autopsy IL. Languages far removed from CMS-2M (such as SNOBOL or APL) would require large additions to the IL and are therefore not suitable for expansion of the ADT.

The Direct Translator is designed to handle large (10,000 line) compilation units by keeping the result of each pass on disk in a special disk format tree notation.

Status of the Autopsy Direct Translator

The ADT performs syntax-directed translation of arbitrary CMS-2M programs into Ada. Parts of the program that cannot be translated automatically are so marked (when known).

The system became operational on small programs in July, 1979. It was expanded to handle large programs in the fall of 1979.

Examples of CMS-2M statements with corresponding Ada translations are given in Figure 8.1.

Assignment:

```
SET SPPK(0,ENTRYC) TO SPPK(0,ENTRYC) + 1 $
```
CMS-2M

```
SPPK(0).ENTRYC := SPPK(0).ENTRYC+1;
```
Ada

If:

```
IF SPINDEX GTEQ COIXMAX THEN
   SET SPINDEX TO 0 $
```
CMS-2M

```
if SPINDEX>=COIXMAX then
  SPINDEX := 0;
end if;
```
Ada

Loop:

```
VARY COWORK FROM COWORK THRU 0 BY -COLOOP $
```
CMS-2M

```
while COWORK>=0 loop COWORK := COWORK-COLOOP;
end loop;
```
Ada

Procedure Declaration:

```
PROCEDURE COMSTUBP INPUT CORAD(SPPK) $
```
CMS-2M

```
procedure COMSTUBP(SPPK : in out SPPKtype) is
```
Ada

Variable Declaration:

```
VRBL SPINDEX I 16 U P 0 $
```
CMS-2M

```
SPINDEX : INTEGER range 0..2**16-1 := 0;
```
Ada

Table declaration

```
TABLE SPCYCLE V 1 100 $
   FIELD ID     I 16 U 0 15 P 0 $
END-TABLE SPCYCLE $
```
CMS-2M

```
type SPCYCLErecordtype is record
   ID : INTEGER range 0..2**16-1 := 0;
end record;
SPCYCLE : array (0..99) of SPCYCLErecordtype;
```
Ada

Figure 8.1. Sample statement translations

The ADT is written to be as language independent as possible. To add a new language, such as JOVIAL, would require only a new parser and a new Step2. The tree format used by the system is compatible with the attribute grammar processing system.

Table 8.1 shows statistics about the running time of the various components of the system. It can be seen that over half the time is spent parsing (and most of that time is taken up by the lexical analyzer). Table 8.2 shows the space taken up by the program.

The system consists of 5K lines of source code. Table 8.3 shows the relative sizes of the five forms of the translated program. It is interesting to note that the original CMS-2M program and the translated Ada result are about the same size. The program as represented by the IL is a little more than half this size.

Table 8.1. ADT program timing statistics

	Seconds/Card	Percent	CONSes/Card	Percent
Parser	.17	57	117	58
Step2	.05	17	38	19
Step3	.04	13	19	9
Prettyprinter	.04	13	28	14
Total	.30	100	202	100
(200 cards/minute)				

Table 8.2. ADT program space statistics

	Machine Words	Percent	Source Lines	Percent
Interlisp	(149634)			
Pattern Matcher	(2907)			
APPSS	2002	10	324	6
Parser	2286	11	742	15
Parser Tables	9373	45	1334	26
Step2	4188	20	1620	31
Step3	2045	10	999	19
Prettyprinter	787	4	167	3
Total	20681	100	5186	100
	(173222)			

Table 8.3. ADT data space statistics

	Lines/Source Line
CMS-2M Source	1.00
CMS-2M Parse Tree	2.25
IL Tree	.56
Ada Parse Tree	1.25
Ada Source	.97

Planned Improvements

The current direct translator does not go far beyond transliteration. To become better, it must become smarter. The use of flow analysis is a step in this direction, allowing us to make some non-obvious translations that go beyond transliteration.

Flow analysis comes in two flavors: control flow analysis and data flow analysis. Control flow analysis has two subparts: the call graph (as in Interlisp Masterscope) and the control flow graph. Data flow analysis also has two subparts: intraprocedural and interprocedural [11, 19].

The following lists some of the things we can do with flow analysis:

1. Range analysis - identify the ranges actually used by scalar variables and refine their declarations to be that range [10].

2. Incorporate local variables into procedures.

3. Remove GO TO statements automatically - transform the program [12], using node splitting for irreducible flow graphs (a rare event in real programs [14]). Ada prohibits certain GOTOs allowed in CMS-2M. These must be eliminated by program transformation.

4. FUNCTION identification - identify normal/abnormal functions for the Ada distinction between functions and value returning procedures.

5. Optimize use of FOR loops - use FOR loop even when limit and increment are not constant, but only unchanged during the loop.

6. Extended error analysis - locate potential errors that a compiler might miss, such as unreachable code that may or may not be meant as a conditional compilation.

Open Problems

There are a number of very difficult problems that arise when translating from CMS-2M to Ada. We do not anticipate that these will ever be solved automatically in the ADT.

Direct Code

The use of assembly language ("direct code" in CMS-2M parlance) intermixed with CMS-2M programs causes serious translation problems. Even assuming that the translated program is to run on the same machine, interactions with the compiler remain:

- In CMS-2M, if direct code refers to an undefined name, that name is implicitly imported.

- The SYSTEM INDEX and LOCAL INDEX features of CMS-2M affect the allocation of registers by the compiler. It cannot be assumed that the Ada compiler will be able to duplicate this.

- Direct code may make assumptions about the storage layout of variables that are not required logically by the program text but are merely compiler artifacts.

- CMS-2M permits the user to write microcode. No attempt is made to translate this.

Arithmetic

As it is currently defined Ada does not permit precise duplication of the arithmetic in CMS-2M programs [8]. If the definition of Ada does not change, translation of arithmetic computations so as to guarantee identical results will be difficult.

Another problem is that CMS-2M has the feature CMODE FLOAT that causes the program to calculate all "intermediate results" in floating point. The meaning of this is obscure.

Miscellaneous Problems

The side effect of the time taken by a piece of code is usually ignored in translation. If it is important (as in a busy wait), the translation may fail to duplicate it. The problem here is identifying time critical code segments, as the constructs of CMS-2M give no clue to this.

CMS-2M permits extensive conditional compilation, whereas Ada allows only conditional compilation of executable code. It is hoped that APSE (Ada Programming Support Environment) will make provision for this [1].

Finally, there remains the problem of "undefined vs. undocumented," i.e., both CMS-2M and Ada claim certain things are undefined, yet the compiler must pick something. This "something" then becomes an undocumented "feature" and programs will (perhaps unwittingly) be written that depend on it. This is really a problem of language definition.

FORMAL SEMANTIC DEFINITIONS

Formal Definitions of Programming Language Semantics

Formal semantic definitions play an important role in the design, specification, implementation, and evaluation of programming languages. Major applications of such formal definitions include

1. standard definition of a language,
2. basis for evaluating and validating a language design,
3. guide to and validation of a language implementation,
4. theoretical basis for reasoning about a language and its programs,
5. basis for automatic or semi-automatic translation of programs in one language to equivalent programs in other languages.

The Autopsy Project is concerned with the first, second, and fifth of these applications.

There are three principal (not necessarily mutually exclusive) styles of formal semantic definitions:

Operational:	The meaning of a language construct is the result of "interpreting" it on a "machine."
Syntax-Directed:	The meaning of a language construct is the result of (recursively) mapping its "syntax tree" into a corresponding semantic object, such as a function denotation.
Axiomatic:	The meaning of a language construct is expressed as an axiom, inference rule, or theorem in a logical calculus.

Syntax-Directed Formal Semantic Definitions

The style of formal semantic definition most useful to the goals of the Autopsy Project is syntax-directed, of which there are two principal methods, differing in style but not in essential content. These methods are denotational semantics, developed primarily by the Programming Research Group at Oxford University [21, 22, 9], and attribute grammars, conceived by Donald Knuth [13, 15].

Denotational Semantics

The denotational method of syntax-directed semantic definition is intended to provide a means for "abstractly" defining a programming language by providing only "essential" characteristics of the language, independent of any details of representation or implementation of language features. In practice, this ideal has been largely, although not totally, realized.

Denotational semantic definitions use abstract syntactic specifications containing only those syntactic features necessary to control the accompanying semantic specifications. In particular, no syntactic features to ensure unambiguous or efficient parsing are normally present in an abstract syntactic specification. A denotational definition must also contain or refer to a well-defined domain of semantic objects, called a semantic universe. In "standard" denotational definitions, this semantic universe is usually a family of ordered domains of values, which represent values computed, or storage transformations effected, by language constructs. On the other hand, in "non-standard" definitions [7], the semantic universe could be a domain of machine language programs (in a denotational compiler specification) [2], or a domain of abstract syntax trees with embedded error messages (in a denotational specification of type checking) [5]. The connection between the syntactic and semantic specifications in a denotational definition is given by a recursive definition of a family of functions that map abstract syntactic structures (conveniently thought of as representing trees) into corresponding semantic objects. This will be explained in more detail in Section 4.3 below.

Attribute Grammars

Attribute grammars [13, 15] are a means for specifying syntax-directed semantic definitions in a way very similar to, but slightly different from, the denotational technique. Traditionally, attribute grammars have used "concrete" syntax specifications (i.e., unambigous grammars that also permit efficient parsing, such as LR(k) grammars), but there is no reason not to use abstract syntactic specifications instead. Indeed, it is advisable to use abstract syntax specifications because they are more compact and contain all syntactic information essential to the definition.

In the literature, the semantic domains of an attribute grammar definition are left "open," and a wide choice is possible, just as in the case of denotational definitions. In fact, even when a specific attribute grammar definition is given, its semantic domains are often implicit from the context of the definition, a situation not permitted in denotational definitions. A recent formal definition of the attribute grammar method of definition itself [3] requires all semantic domains to be explicitly specified, a more satisfactory state of affairs.

Most clearly distinguishing the denotational and attribute grammar methods of syntax-directed semantic definition is the mechanism used to associate the syntactic and semantic parts of a definition. Attribute grammars associate "attributes" with each nonterminal of the syntactic specification. These attributes are intended to represent certain semantic information about the syntax tree rooted with the corresponding nonterminal. Sets of equations define their values, one set associated with each production, relating the values of the attributes of the nonterminals appearing in that production. The sets of equations, and therefore the attributes, associated with each instance of each production in a syntax tree are linked together into one large system of equations in a manner corresponding to the interconnection of the productions in the tree, as defined in [3]. The interdependency between the attributes associated with the nonterminal nodes in a syntax tree can be viewed as representing a certain semantic "information flow" among attributes, which leads in many cases to a very intuitive formulation of syntax-directed semantic definitions. Denotational definitions can in fact be written as (specialized) attribute grammars.

Tools

Tools exist to enable denotational and attribute grammar styles of semantic definitions to be input to and processed by computers. Two such tools will be discussed in this section.

SIS

Denotational definitions can be specified, implemented, and tested using the Semantic Implementation System (SIS), developed by Peter Mosses at Oxford University and later at Aarhus [16, 17]. SIS provides means for specifying (1) concrete and abstract syntax and the precise computable relationship between them, in a language GRAM, and (2) syntax-directed denotational semantic equations, in a language DSL (Denotational Semantics Language). These syntactic and semantic specifications and the languages in which they are expressed are supported by a third language and its interpreter, LAMB, an enriched form of lambda calculus. Both the syntax and semantic equation specifications are translated into equivalent versions in LAMB, the first (syntax) into SLR(1) parser tables [4] and the second (semantics) into a LAMB function that maps abstract syntax trees into corresponding semantic objects, in this case represented by LAMB denotations.

AGAPE

Attribute grammar definitions can be specified, implemented, and tested using the system AGAPE (Attribute GrAmmar Processing systEm), developed by Jeffrey V. Cook at ISI. AGAPE will be more fully described in Section 5, but a brief description will be given here. AGAPE, like SIS, provides means for specifying syntax and corresponding semantics. Concrete syntax (which must be SLR(1)) is specified using a (LISP-like) notation for context-free productions, and corresponding sets of semantic equations are similarly specified. Abstract syntax plays no direct role in AGAPE, but its inclusion is a likely future enhancement. The AGAPE system is supported by Interlisp, and the specifications expressed are translated into equivalent Interlisp functions. Consequently, all "primitive" semantic functions that appear in semantic equations must be given Interlisp definitions. The values of attributes over a (concrete) syntax tree are computed by a tree-traversing automaton evaluator, which in effect solves the semantic equations. At present, only nonrecursive (noncircular) equations are handled by AGAPE,

but the symbolic solution of recursive semantic equations using fixed point operators is a planned future enhancement.

AGAPE (unlike SIS) has facilities for performing certain checks on a definition prior to actually "running" it. The semantic equations are checked to ensure that all the attributes are defined in a consistent manner. Further, AGAPE computes the dependencies between attributes so that the designer of the definition can determine whether his intentions have been realized. The attribute dependency test also provides a basis for determining whether the definition is "circular" (i.e., recursive). Since semantic equations in AGAPE are in effect LISP S-expressions, the functions in these equations are not typed, and therefore no type checking is done by AGAPE.

Components of Syntax-Directed Formal Semantic Specifications

The major components of syntax-directed semantic definitions will be discussed in this section. To begin with, concrete and abstract syntax definitions and simple ways to specify the relation between them will be illustrated. Then semantic equations will be illustrated for both denotational and attribute grammar definitions.

Syntax Definitions

Concrete syntax definitions are usually specified by means of a context-free grammar in which the productions are designed to be unambiguous and to facilitate efficient parsing. Often this kind of grammar, although quite practical, introduces syntactic structure that is not essential to the syntax-directed definition. A corresponding abstract syntax definition, which controls the semantics, is more desirable. A simple example that illustrates these concepts in simple syntax specifications for expressions is given below.

CONCRETE SYNTAX

```
expr    ::= expr + term
          | - term
          | term                    : term

term    ::= term * factor
          | factor                  : factor
```

```
factor   ::= primary ** factor
           | primary                    : primary

primary  ::= id
           | ( expr )                   : expr
```

(Syntactic Domain Declarations) -- expr, term, factor, primary: Expr

The items ": term", ": factor", ": primary", and ": expr" and the syntactic domain declarations are not part of the concrete syntax; their purpose will be explained shortly. The nonterminals term, factor, and primary are used to make the above grammar unambiguous and SLR(1). A corresponding (context-free) abstract syntax for the same expressions is given below.

ABSTRACT SYNTAX

```
expr   ::=  expr + expr
         |  - expr
         |  expr * expr
         |  expr ** expr
         |  id
```

The abstract syntax is clearly ambiguous. The concrete and abstract syntax trees of the expression A*(B+C)**D are shown in Figure 8.2. It is clear that many "inessential" levels of syntax have been eliminated in the abstract syntax tree. The linkage between the concrete and abstract trees was established by the items ": term", etc., and the syntactic domain declarations. For example, when a production expr -> term is applied in the concrete syntax tree, the item ": term" in the concrete syntax specification decrees that the node labeled "term" will not be present in the corresponding abstract syntax tree. Similar remarks hold for the productions term -> factor, factor -> primary, and primary -> (expr). The syntactic domain declarations specify that all the remaining nodes labeled "expr", "term", "factor", and "primary" in the abstract syntax tree shall be labeled "expr". The manner in which these node deletion and relabeling directives were included in the concrete syntax specification is a standard feature of SIS. SIS converts such a specification into tables for an SLR(1) parser and a function that, during parsing, produces a corresponding abstract syntax tree.

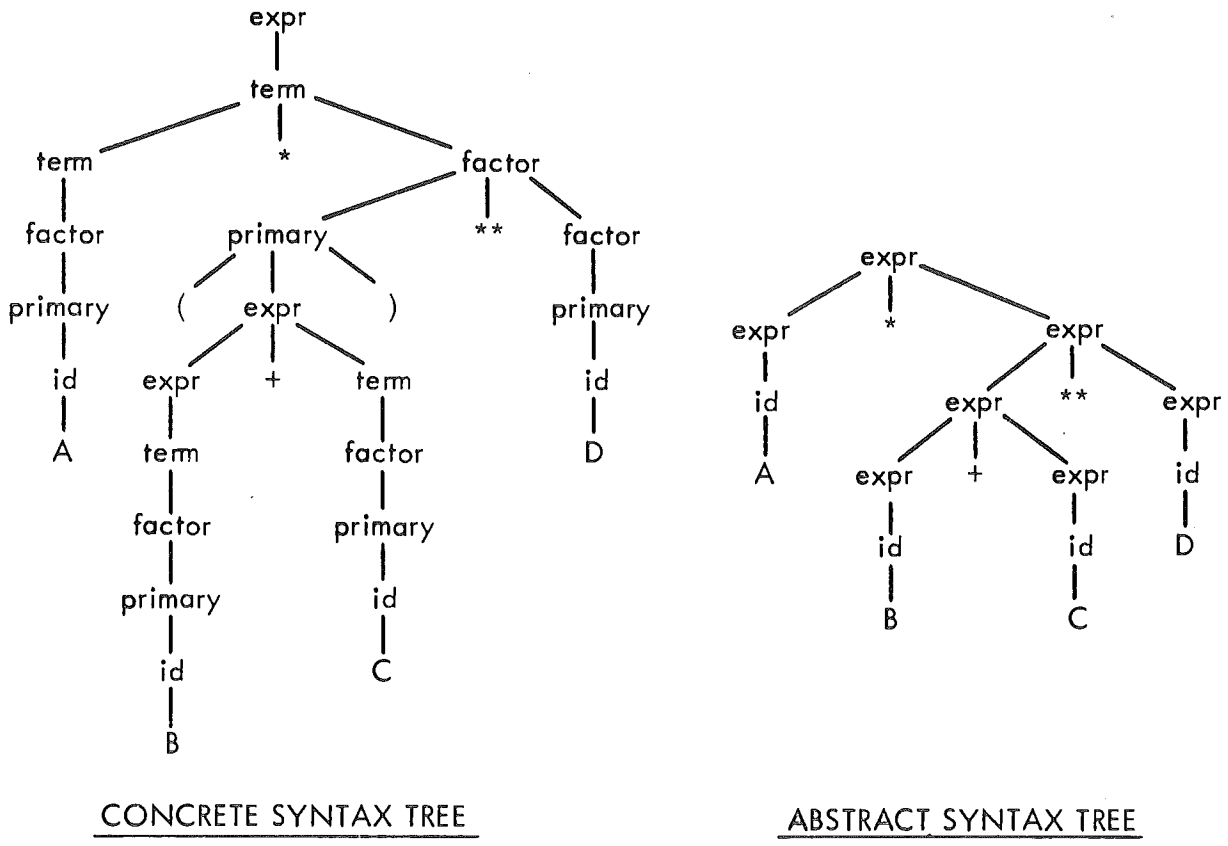


Figure 8.2. Concrete and abstract syntax trees of A*(B+C)**D

Semantic Specifications

The semantic specification in denotational definitions consists of the definition of recursive functions that map abstract syntax structures into corresponding semantic objects. For example, let Stmt denote the domain of abstract syntax trees of statements in some programming language L, let S denote the domain of "abstract stores" (where an abstract store records the current value of each variable in an L-program), and let [S -> S] denote the domain of (continuous) functions from stores to stores. Then the mapping

Ms: Stmt -> [S -> S]

gives the semantics of L-statements as store-to-store transformations. The following

illustrates how the semantics of statements is syntax-directed: Suppose that L-statements can also be statement sequences via the (ambiguous) production

stmt -> stmt ; stmt .

The semantic mapping Ms would be "extended" to statement sequences (which are also in the syntactic domain Stmt) via the recursive definition

Ms[stmt1 ; stmt2](s) = Ms[stmt2](Ms[stmt1](s)) .

This states that the semantics of the "compound" statement stmt1;stmt2 is the indicated composition of the semantics of the statements stmt1 and stmt2.

The semantic specification part of attribute grammar definitions is given by a set of equations associated with each production of the syntactic specification (whether concrete or abstract). These equations define the value of the attributes appearing in them, in a manner precisely defined in [3]. The above example can be repeated as an attribute grammar fragment by letting the nonterminal stmt have an inherited attribute sin (input state) and a synthesized attribute sout (output state). Letting, for example, sin.stmt be a variable that denotes attribute sin of nonterminal stmt, the attribute grammar fragment associated with the abstract syntax production for compound statements is

$$\begin{aligned} \text{stmt} ::= \text{stmt}_1 \text{ ; } \text{stmt}_2 \qquad & \text{sin.stmt}_1 = \text{sin.stmt} \\ & \text{sin.stmt}_2 = \text{sout.stmt}_1 \\ & \text{sout.stmt} = \text{sout.stmt}_2 \end{aligned}$$

The occurrences of "stmt" in the right part of the above production are subscripted to identify corresponding occurrences in the semantic equations. The above semantic equations lead to the "attribute dependency graph" shown in Figure 8.3. If the attributes sin and sout are regarded as conveying semantic information, then this graph clearly depicts the semantic "information flow" through the immediate constituents of compound statements.

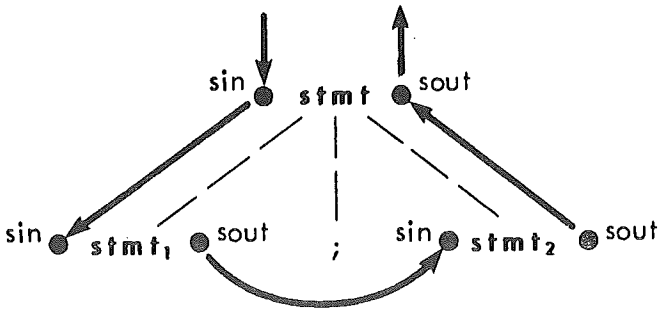


Figure 8.3. Attribute dependency graph

Using a method given in [3], the above attribute grammar fragment can be converted into an equivalent fragment that uses only synthesized attributes. Letting "state" be a synthesized attribute of stmt, where state ranges over [S -> S], the above fragment can be converted to the equivalent

$$stmt ::= stmt_1 ; stmt_2 \qquad state.stmt(s) = state.stmt_2(state.stmt_1(s))$$

The correspondence between the above fragment and its denotational counterpart is unmistakable.

Denotational Semantic Definition of CMS-2M

A principal use of formal semantic definitions in the Autopsy project is to precisely define the source and target languages of the semi-automatic translations being studied. As an initial experiment, a denotational definition of the source language CMS-2M is being formulated. At the time this exercise was initiated, the system AGAPE was far less complete than SIS. It was therefore decided to code and test the definition in SIS.

CMS-2M

CMS-2M is a late-1950s-vintage high level programming language used by the Navy. It is not a block-structured language in the ALGOL tradition, but has more of the flavor of FORTRAN. CMS-2M programs are organized into SYSTEMs that are separately compiled.

Each SYSTEM contains system data blocks (SYS-DDs), which are devoted exclusively to data unit declarations, and system procedure blocks (SYS-PROCs), which contain only procedure declarations. All data and procedures must be declared before use in each SYSTEM. All external references between the separately compiled SYSTEMs are resolved by a linking loader.

CMS-2M has a complement of more or less standard data types, including integer, fixed and floating point, Boolean, Hollerith (character), arrays and records, label switches, and procedure switches. CMS-2M has an elementary type of pointer called a "core address," the address of a simple variable, or the base address of an array or subarray. A core address is obtained by applying the intrinsic function CORAD to a variable. CMS-2M has fairly standard expressions and statements, the latter including the usual assignment, conditional, iteration, procedure call, goto, and computed goto and procedure call (use of label and procedure switches). Provision is also made for including segments written in assembly language ("direct" code), which in the case of CMS-2M is for the AN/UYK-20 computer. CMS-2M has no intrinsic input-output statements, and input-output is accomplished by direct code.

CMS-2M is unusual because among a collection of SYSTEMs (called a "compile") there is no identifiable "main" program or procedure. A procedure is designated as the "starting point" of a CMS-2M compile when its entry point is used to begin execution after the constituent SYSTEMs have been linked and loaded. Therefore the semantics of a CMS-2M compile is the semantics of the collection of procedures contained in it.

Formal Definition of CMS-2M

The denotational definition of CMS-2M has several features in common with other denotational definitions written using the SIS system. It has an SLR(1) concrete syntax specification and the specification of how corresponding abstract syntax trees are to be constructed during parsing. Semantic functions are defined recursively on the abstract syntactic constructs. Following the style of a machine-checked semantic definition of ASPLE [5], the semantic specification is divided into two "passes". The first pass checks prior declaration of items before use and performs type checking. The second pass, which assumes that the first pass has been done, defines the execution ("run time") semantics of the procedures.

The formal definition of CMS-2M is ongoing and will be completed during the coming year.

REFERENCES

1. Buxton, John N., et al., *Requirements for Ada Programming Support Environments*, Department of Defense, Technical Report 3D1079 Pentagon, February 1980.

2. Chirica, Laurian M., *Contributions to Compiler Correctness*, UCLA Computer Science Department, UCLA-ENG-7697, October 1976.

3. Chirica, L. M., and D. F. Martin, "An Order-Algebraic Definition of Knuthian Semantics," *Mathematical Systems Theory* 13 (1), 1979, 1-27.

4. DeRemer, F. L., "Simple LR(k) grammars," *Communications of the ACM* 14 (7), 1971, 453-360.

5. Donzeau-Gouge, V., G. Kahn, and B. Lang, *A Complete Machine-Checked Definition of a Simple Programming Language Using Denotational Semantics*, IRIA Laboria, Technical Report 330, October 1978.

6. Donzeau-Gouge, V., G. Kahn, and B. Lang, *Formal Definition of Ada*, Honeywell, Inc. and CII-Honeywell Bull, 1980.

7. Donzeau-Gouge, V., *Utilisation de la Semantique Denotationnelle pour l'Etude d'Interpretations Non-Standard*, IRIA Laboria, Technical Report 273, January 1978.

8. Gillmann, Richard, "Living with Variations in Machine Arithmetic," in *Ada Test and Evaluation Workshop*, Defense Advanced Research Projects Agency, October 1979.

9. Gordon, Michael J. C., *The Denotational Description of Programming Languages: An Introduction*, Springer-Verlag, 1979.

10. Harrison, W., *Compiler Analysis of the Value Ranges of Variables*, IBM T. J. Watson Research Center, Research Report RC-5544, 1975.

11. Hecht, Matthew S., *Flow Analysis of Computer Programs*, North-Holland, 1977.

12. Knuth, D. E., and R. W. Floyd, "Notes on Avoiding 'GO TO' Statements," *Information Processing Letters* 1 (1), 1971, pp. 23-31.

13. Knuth, D. E., "Semantics of Context-Free Languages," *Mathematical Systems Theory* 2 (2), 1968, 127-145.

14. Knuth, D. E., "An Empirical Study of FORTRAN Programs," *Software Practice and Experience* 1 (12), 1971, 105-134.

15. Knuth, D. E., "Semantics of Context-Free Languages: Correction," *Mathematical Systems Theory* 4 (1), 1971, 95-96.

16. Mosses, Peter D., *Mathematical Semantics and Compiler Generation*, Ph.D. thesis, University of Oxford, 1975.

17. Mosses, Peter D., *SIS: A Compiler-Generator System Using Denotational Semantics (Reference Manual)*, University of Aarhus, Institute of Mathematics, Department of Computer Science, DK-8000 Aarhus C, Denmark, 1978.

18. Samet, Hanan, "Proving the Correctness of Heuristically Optimized Code," *Communications of the ACM* 21 (7), July 1978, 570-582.

19. Schaefer, Marvin, *A Mathematical Theory of Global Program Optimization*, Prentice-Hall, 1973.

20. Schatz, Bruce R., Bruce W. Leverett, Joseph M. Newcomer, Andrew H. Reiner, and William A. Wulf, *$TCOL_{Ada}$: An Intermediate Representation for the DOD Standard Programming Language*, Carnegie-Mellon University, Computer Science Department, CMU-CS-79-112, 1979.

21. Scott, D. S., and C. Strachey, "Toward a Mathematical Semantics of Computer Languages," in Fox, J. (ed.), *Proceedings of the Symposium on Computers and Automata*, pp. 19-46, Polytechnic Institute of Brooklyn Press, 1971.

22. Stoy, Joseph E., *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*, The MIT Press, 1977.

9. PORTABLE PACKET RADIO TERMINAL

Research Staff:
Tom Ellis
Steve Saunders

Consultant:
Bob Parker

PROBLEM BEING SOLVED

ARPA is currently developing a Packet Radio Network (PRNet) that will provide a wideband data communication capability much like the ARPANET but with the added dimensions of mobility and dynamic configurability. As this concept gains acceptance in the military services, fundamental choices will need to be made about mobile terminals for use with the system.

The ISI Portable Packet Radio Terminal Project, begun in June 1978, addresses the terminal characteristics desirable at the user level, the kinds of interfaces and system architectures required to support them, and the impact of such issues on terminal designs and other portions of the system. The work is intended to result in a demonstration-level portable terminal (PRT) to test and evaluate various solutions to the problems raised by extreme portability in the packet radio environment.

We feel that a comprehensive scientific study of these issues is important for the following reasons:

- To properly exploit, match, and couple terminal design to the communications attributes and special capabilities of the PRNet.

- To determine those user interface characteristics best suited to supporting the major user needs in a mobile tactical environment.

GOALS AND APPROACH

This project centers on the design of a display terminal with voice, keyboard, and stylus or pointing input. We envision a mixed-media terminal capable of at least the following: text input and output, graphics input and output, and two-way voice communication.

Extreme Portability

Although in commercial parlance today "portable" may mean only that the manufacturer chose to attach a handle, we mean something more specific: a portable terminal is one

that can be conveniently carried and used in the performance of some other--primary--job. That is, it must be small enough to be carried in a pocket or on a strap, out of the user's way while he works, yet convenient when he needs it. We believe that this level of portability is essential to the full exploitation of the potential of packet radio.

Not every terminal or element of the PRNet needs to be so portable, of course. Large host processors, databases of messages and maps, and high power repeater units cannot be. Some terminals will be mounted in vehicles or buildings and need not be pocket sized. But we are convinced that there is a need for fully portable units as well, and that it is feasible to build them.

Simple Interfaces

One requirement of a useful portable terminal is that its design support simple, straightforward user interfaces for a wide variety of applications. Suitable interfaces must be natural for use as non-exclusive tools in doing a job; operation of the terminal must not require more training and practice than the user's main task. The implementation of such smooth user interfaces will in many cases require a more powerful mechanism, more computation or communication or display capability than conventional interfaces would use. We are trying to discover how best to provide enough power for some interesting classes of interfaces in a convenient and low-cost device.

Design Constraints

If the PRT is to be extremely portable, it must not be too large, too oddly shaped for convenient carrying, or too heavy. We have shown a mockup for a PRT that we believe is a reasonable size; it is about 7 by 4 by 1.5 inches and weighs about two pounds. This size and shape allow it to be carried in a large pocket. The volume is sufficient to contain a terminal, radio unit, and batteries; a lid opens to give more surface area for display, keyboard, and operating controls.

A more equidimensional shape with the same volume, while perhaps making radio construction simpler, would be too bulky for any pocket and harder to open for use; a much flatter shape, while avoiding a hinged display section, would be too wide for convenient carrying in a pocket. We are continuing to explore these options.

A highly portable device must carry its own power source, rather than relying on wires. Today this implies batteries. One of the most stringent and critical limitations on what can be put in a PRT-size package arises from the available battery energy density. Low power circuits and design techniques, such as low duty cycle operation of subsystems, will be strongly favored in all design tradeoffs.

In order to be *used* and not just carried, the PRT must be simple in its operational controls. We believe that the terminal should have few switches, lights, and knobs and that nearly all control functions should be accomplished via the keyboard and display. The terminal should appear to the user as simple as the application allows. We realize that the mechanisms that underlie and support this simplicity may be very complex, but good design will insure that users need never realize it.

PROGRESS--Issues Explored and Reported

We have written, and continue to expand, a comprehensive document exploring many of the issues surrounding the design and operation of the PRT [2]. The document presently includes the following chapters:

- Multimedia Synchronization
- Communication Data Rates
- Touchpanels and Pointing Devices
- Graphics--Display Capabilities, Representations, Interaction.

We have written a shorter functional specification document giving the conclusions and key arguments of the issues paper, without the fully detailed exploration of alternatives that the issues paper provides [3]. This specification should be useful to ARPA and others in defining and developing the radio unit for the PRT and in considering the wider implications of extremely portable and extremely capable tactical communications.

Multimedia Synchronization

We have identified a problem that did not exist for previous communications systems but arises only in the context of simultaneous multimedia connections: the problem of synchronization. It is caused by the widely differing data rates and delay or burstiness characteristics of the media we expect to employ: speech, text, and graphic images.

The synchronization problem is the risk that users of simultaneous multiple media such as speech and graphics may be fooled by the different delays involved. A voice or text reference to a picture must be in the presence of that picture, not the one before or after. (For example, the messages "Place your battery at the point marked X" and "Fire at the point marked X" had better be in the presence of the right maps.) Difficulties in assuring synchronization come about because speech is continuous, with low bounded delay in transmission, while sending a new graphic image may take several seconds. But users who are not computer communication experts should not be expected to appreciate this difference or to tailor their actions to it. Some support from the communication system is required.

We have brought the synchronization problem to the attention of the Internet Working Group and ISI's NSC project as the cognizant technical groups who are closely involved with the protocols and mechanisms needed to provide a solution.

PROGRESS--HARDWARE DEVELOPMENTS

We have been exploring the available technologies to identify and pursue developments in those that promise to be useful in the low power, highly compact world of the PRT. The key requirements are for display, graphic and keyboard input, processing, and memory. Packet radio unit miniaturization is being pursued elsewhere, and we expect to use the results of these efforts as they become available.

Display

A major component of our efforts has been to provide a shaped push to the necessary display technology. Electrophoretic (EP) displays have a unique combination of characteristics that make them the technology of choice for a battery-powered high-resolution display. EP displays offer extremely low power consumption, high contrast, and very wide viewing angle.

EP materials. At the start of this project, there were some uncertainties about the EP material, a suspension of light-scattering pigment particles in a dyed dielectric liquid. Through our subcontract with Xerox Palo Alto Research Center we have cleared up many of these doubts.

The material supplied to us by Xerox has been entirely stable for a year now in operational display cells and in storage. The operating speed of our test cells has been in the neighborhood of 15 milliseconds, a significant improvement over the 100 milliseconds available when we began to investigate EP. Our cells also have much lower switching voltage requirements of 25 to 40 volts compared to the 100 volts that we expected. There was doubt about the compatibility of the EP material with commonly available photosensitive polymer materials used for internal cell structures; we have seen no adverse interactions in the cells we have built using these materials. We have enough Xerox EP material still on hand to continue with display experimentation.

Particle migration. Another result of the subcontract with Xerox was the development of a concept for isolating adjacent pixels in an EP display to prevent the migration of pigment particles from active regions to inactive ones. This was a significant concern; a pixel could become completely depopulated of EP particles in as few as five on-off cycles.

The solution is to provide barrier walls within the cell in a grid or "waffle" pattern, separating each pixel from its neighbors. The walls need not extend to the front surface of the display or entirely seal off each pixel; walls that leave a gap of up to a third of the cell thickness seem to be sufficient to prevent migration and degradation of display performance. The grid can be made of photoresist film by standard photolithographic techniques.

Cell structures. Using the EP material supplied by Xerox, we have explored a variety of materials and techniques for building and filling cells. We have used NESA and NESATRON conductively-coated glass as cell fronts and backs; MYLAR polyester sheet and RISTON photopolymer film as spacers; heat-sealed RISTON, epoxy glue, and hot-melt adhesive film to assemble the cells; holes drilled in the glass and a no-hole open-faced technique to fill the cells with the EP material. We built cells using a block of fine parallel copper wires embedded in epoxy to explore a way of gaining rear (through substrate) access to display pixels. We developed a structure for the cell sealer and spacer which allow a bellows-like action so that bubbles can be avoided or pumped out during the filling process. We built a total of about fifteen cells, all of which exhibited correct EP display behavior.

Thin film transistor arrays. Because EP material has almost no inherent threshold effect, it needs a nonlinear element at each pixel in order to function in an X-Y matrix array. Thin-film transistors (TFTs) appear to be the most promising technology for providing this capability over large area displays. TFTs offer the additional capabilities of amplification (allowing logic-compatible low voltages on the X and Y select lines to control the higher drive voltage of the display), integrated capacitive storage at each pixel (allowing writing to proceed much faster than the EP particle migration), and the potential for full integration of scanning and driving electronics on the display substrate itself.

Our subcontract with Panel Displays Incorporated (PDI) to do development work in TFT arrays has produced less spectacular results than the Xerox EP material. We had some early test arrays laid out by PDI and fabricated by another thin-film deposition laboratory; these did not have the TFT characteristics needed to drive the EP arrays. Standard commercial processing (set up to produce optical coatings and the like) apparently does not permit the special conditions and control necessary to make TFTs of high quality. We have been awaiting the completion of new thin film deposition equipment at PDI that will have the necessary control and flexibility.

Display mosaic. Display organization for the PRT offers some interesting possibilities for innovation. PDI has proposed that the TFT active matrix could be segmented into autonomous tiles, each one comprising an entire logical display, allowing arbitrarily large displays to be built as a mosaic of these tiles. This is an especially promising direction for early work since it allows the tiles to be made smaller, an advantage with expected initial defect statistics. For the PRT display, tiles of 128x128 pixels would probably give significant fabrication advantages and would allow access to each tile at an edge of the display. Larger mosaics of smaller display tiles would require access to the rear of at least some tiles; this would imply considerably more development effort.

Discrete transistor arrays. We are continuing to consider alternatives to the TFT array for the PRT display; we built the wire-block cells mentioned above as part of this ongoing exploration. We have defined several alternative concepts as fallback positions in case TFT arrays do not become available in time.

One such concept is to make substrates with interconnecting circuitry out of thin films (a well-established commercial technology) and insert a discrete transistor chip at each pixel location. This gives new meaning to the term "semiconductor dice": since standard silicon wafers are almost as thick as the ten mil pixel spacing we require, the individual transistors would be very nearly cubes. A waffle pattern would be put down to locate the dice, which could be made with a symmetric pattern of base and emitter contacts to allow arbitrary rotation during placement. We note that Sanyo has built a TV display out of 38400 discrete LEDs using similar techniques.

Back-access substrates. Another alternative is to build substrates that give access through their thickness so that we can put the display medium on one side and arbitrary circuitry on the other. The wire-block idea is one way to do this. If the wires are small enough (roughly speaking, less than a fourth of the pixel spacing in diameter) and parallel enough, each pixel contact pad on one side of the block is connected to a well defined area on the other surface, but isolated from its neighbors. We built wire blocks by winding fine magnet wire into a coil while keeping it wet with epoxy, and slicing the coil radially after the epoxy hardened. The resulting display cells allowed us to write and paint images in the EP medium using a stylus held at the writing voltage. The low power requirements and high resolution capability of EP were pointedly demonstrated by the appearance of a fingerprint in the display when one of us touched the wire surface while holding the current-limited write-voltage electrode in the other hand! Unfortunately, the wire blocks that we made turned out to be slightly porous, allowing the EP material to seep gradually into the substrate.

Another way to build substrates with connections through them is to stack up strips of substrate material having crosswise conductive stripes on them. This would allow the controlled incorporation of element storage capacitors into the substrate itself. If it turns out that defect-free yield of TFTs over large areas is a problem, the necessary TFT circuitry might be built on such strips and tested a line at a time rather than a whole tile at a time. The strips could be manufactured as a continuous ribbon, inspected for defect-free segments of sufficient lengths, and cut for stacking and bonding together.

Homogeneous isotropic connectors. To make use of connections on the back of a substrate we need a way to mate the array of contact points or pads with a similar array on another substrate. What is required is a slightly resilient conductor between each

corresponding pair of contact pads and insulation to separate these conductors from each other.

An electrically anisotropic material having lower resistivity in one direction than in others could serve as both through connection and lateral insulation. Some such materials are in commercial production, but they are too coarse-grained for our uses.

We have discovered that in the range of dimensions we need, an electrically isotropic material can be both conductor and insulator in a homogeneous (single-component) connector. The critical ratio of leakage resistance to connection resistance depends only on the geometry of the connection: pad size, pad spacing, connector thickness, and pad-to-pad misalignment. The actual resistances are scaled by the bulk resistivity of the connector material.

We have constructed a mathematical model that yields, for a given required misalignment tolerance, the optimum pad size to spacing ratio. One can then choose a connector thickness to meet the leakage resistance ratio requirement and a material to give the required resistance values. For one display arrangement we found that shellac has the right resistivity, and that a one-mil thickness would give a leakage resistance ratio of better than 1000.

Touchpanel--Crossed Resistive Sheets

There is a very simple way to make a touch-sensitive panel or surface that can be used for computer input as a pointing, sketching, or drawing device. The idea is to mount two sheets of uniformly resistive material, at least one of them flexible, facing each other with a very small air space between them. Pressure at some point on the flexible sheet causes a point or small area contact between the sheets. If a voltage is applied between opposite edges of one sheet, the other sheet will read out the position (along the voltage gradient direction) of the contact. Then if the roles are reversed, and a voltage is placed on the second sheet perpendicular to its direction in the first, the other coordinate of the contact is available. Thus a very simple physical structure and simple switching suffice for X-Y position location.

This notion has been used and published before [1], but only recently have transparent, uniformly resistive sheet materials become widely and cheaply available.

Both glass and MYLAR sheets with tin oxide or indium tin oxide coatings of better than 5 percent uniformity are now available from several manufacturers. With these materials we can build a touchpanel overlaying a flat display that adds less than 30 mils to the thickness of the display panel.

This crossed resistive sheet touchpanel has several advantages over other pointing devices for the PRT application. Its low volume is one; the rear, rigid surface of the touchpanel can even be incorporated into the front glass of the display cell! It also requires very little power to operate (essentially only that of the analog-to-digital converter required to read out the coordinates). The edge connections in this design are simple continuous contacts, in contrast to some other touchpanel designs using single resistive sheets. No special pointer or stylus or dangling cord is needed; a pencil, a twig, a finger, even a gloved knuckle serve quite well.

The positioning *accuracy* of this kind of touchpanel is fairly poor, in contrast to its essentially infinite *resolution*; it cannot be used for drafting or digitizing of pictures or overlayed drawings. The accuracy is limited by variation in the resistance of the coating over the active area; this variation is normally quite smooth, but significant. However, for use in a smart terminal, where visible cursor feedback is available or where simple pointing to menu items or other displayed objects is wanted, 3 to 5 percent accuracy is fully adequate and attainable. The resolution can be made sufficient to allow a careful user, with a displayed tracking cursor, to pick out a single pixel if he needs to.

The oxide coated plastic film used in these touchpanels is subject to damage if roughly handled before installation; a tiny scratch in the active surface can cause a large deviation in the position indication. However, once the touchpanel is assembled, the sensitive surfaces are mutually protected and sealed; touchpanels made with one rigid side, such as glass, are additionally protected from damage to the coating due to excessive bending of the film. We expect the durability of these touchpanels to be entirely adequate for tactical use.

We have built a prototype touchpanel and an interface to an NLS display terminal, in order to verify and gain experience with the technology. This prototype used coated MYLAR film from Sierracin/Sylmar, and NESA glass from PPG. Its performance as a pointing device is entirely satisfactory. We did not provide the associated buttons that are

needed when using NLS, which assumes a handheld *mouse* as a pointing device; thus the touchpanel is not fully integrated into this pre-existing application.

Touchpanel--Crossed Beams

We originated a different and novel approach to providing a high resolution pointing capability, involving an arrangement very similar to the familiar grid of parallel light beams across the display face. The new idea is to allow each light emitter to illuminate many detectors, and each detector to see many emitters. To prevent confusion, each emitter is pulsed on momentarily in sequence. The resulting combinatorial multiplicity of beams covers the active area quite densely, if the emitters and detectors are correctly arranged, and gives a capability for high-resolution location of any object (such as a fingertip) that is placed on the screen. Considerable computation is required, however, to extract object position information from the raw pattern of beam interruptions. Arrangements of emitters and detectors that give uniformly high-resolution coverage unfortunately make the computations even more complex than simpler layouts. We built computer simulations of several model layouts in order to judge the possibilities, but have abandoned this as a touchpanel technology for the PRT in favor of the crossed resistive sheet design described above.

Programmable and Specialized Keyboards

We expect that application software will make use of a touch-sensitive display to provide the equivalent of keyboards or menus that are rapidly alterable. Any set of symbols or options that become needed, even if only momentarily, can be presented for choice on the display and selected on its touch-sensitive surface.

Given a compact, robust, and cheap touchpanel technology, it seems sensible to implement the terminal's keyboard as another touchpanel rather than as an array of mechanical switches. The standard key arrangement would be printed onto the upper sheet of the panel. We can then suit the keyboard arrangement to the application simply by providing a thin printed overlay for the keyboard area. This allows considerable flexibility in adapting the terminal to various simple application-specific interfaces, when the changes in key assignments are not so rapid as to warrant use of the display surface.

CMOS Microprocessor

Among currently available digital circuit families, only CMOS offers the low power and performance suitable for a battery-powered PRT. We have begun to acquire some experience with CMOS microprocessors and other circuitry by setting up an RCA 1802 microprocessor board, adding a UART and 4K bytes of memory, and building an experimental 16 by 16 display matrix drive I/O circuit. We have rewritten the RCA monitor program to allow data transfer at 9600 baud over the UART, and have operated the 1802 using the ISIB TENEX system for file storage and terminal support.

There are several CMOS microprocessors on the horizon that may be more powerful than the 1802 or otherwise better suited to the PRT task; our benchtop setup has the flexibility to allow us to take advantage of these as they materialize.

IMPACT

We expect this work to influence designs of portable terminals, the applications that use them, and the communications and programming environments that support them. Some specific areas of impact will be on PRNet and Internet protocols, on miniature packet radio units, on tactical terminal application programs, on tactical communication systems and procedures, and in the ergonomics of battlefield and command computation.

We expect a major impact to be effected through a working demonstration model of a truly portable, highly capable, and smoothly usable terminal. The model will serve to raise the expectations of other designers by giving an "existence proof" that higher expectations can be met. It will also serve, we hope, as a spur to further innovations and improvements.

FUTURE WORK--TECHNOLOGY EXPLORATION

Benchtop PRT model

The benchtop 1802 microprocessor system will serve as our first model of the PRT, and allow some experimentation with display interaction and application development. We will augment it as necessary to support this modeling. In particular, we expect to use the TENEX system to simulate the behavior of a PRNet and radio unit. We also plan to add an interim CRT-based simulation of the EP display, one or more touchpanels, more memory, and I/O interfaces.

Press for TFTs

We believe that TFTs remain the technology of choice to provide the matrix selection, storage, and drive functions. In spite of initial delays in getting TFT arrays, we plan to continue pressing for developments in this key area.

Pursue alternate matrices

At the same time, we will continue to seek out other ways to select and drive EP display elements. While we believe that these stopgap methods will eventually be superseded by TFTs, they can perhaps serve the immediate need to get EP displays in operation to explore their uses.

FUTURE WORK--ISSUES EXPLORATION

Refine functional specification

We will continue to refine the functional specifications document. As we consider more scenarios and applications, we will be able to increase the level of detail in the specification of the functionality that will best meet the user needs and support the applications. The continuing advances of electronics and information processing technology will provide additional surprises and opportunities.

Continue issues studies

We will continue to explore and report on the design and operational issues surrounding the use of extremely portable personal packet radio terminals. We plan to add chapters to our issues document covering modes of interaction using the display and touchpanel; authentication, encryption, and key distribution; and a survey of some applications of the PRT with attention to their specific problems and characteristics.

REFERENCES

1. Pobgee, P. J., and J. R. Parks, "Applications of a low-cost graphical input device," in *Proceedings of the IFIP Congress*, 1971.

2. Saunders, S., and T. O. Ellis, Issues in the Portable Packet Radio Terminal (PRT) design (in progress).

3. Saunders, S., and T. O. Ellis, Functional specification of a Portable Packet Radio Terminal (PRT) (in progress).

10. MULTIAPPLICATION SUPPORT TERMINAL

Research Staff:
Louis Gallenson
Alvin Cooperband
Richard Shiffman
Jeff LaCoss
Jerry Wills

Support Staff:
Lisa Holt
Pamela Kaine

PROBLEM BEING SOLVED

The MAST project is a development effort to demonstrate the use of applications partitioned between terminal and host with reduced dependency on host and communications resources. The effort demonstrates how to take advantage of the capabilities of state-of-the-art terminals and examines the issues of application software preparation, user interface, host interface, and ARPANET interface. Special consideration is given to the user and network interface requirements posed by the National Software Works (NSW).

GOALS AND APPROACH

The primary objectives for the MAST hardware are that it be low in cost, flexible, and simple, not only from the perspective of the user but also in interaction with responsible applications, system-level software (NSW, TOPS-20, and TENEX), and network protocols. It should behave like one or more Network Virtual Terminals (NVTs) with no effect on existing programs that expect an NVT interface. Currently the terminal can be characterized as a video screen capable of displaying a full page of text (nominally 60 lines of 80 characters) in multiple fonts and character presentations, a state-of-the-art microprocessor with sufficient memory for a multiapplication environment (128K bytes), firmware to manage the resources, a modern keyboard with several special function keys, and a pointing device (mouse). The terminal as described will be the standard of the near future and is available today from some vendors.

The applications selected for demonstration are an NLS (On-Line System) work station, an NVT, an NSW front end, and a screen editor/formatter suitable for document preparation. The selection was based on need and use at ISI as well as the generation of a realistic, nontrivial set of requirements for the terminal and application-software

development system. The most demanding application is the editor/formatter. The primary objectives of the MAST screen editor are to provide the user with a simpler, more convenient, and efficient low-cost way to prepare and revise documents than is currently available. As much of the editing as possible will be supported directly in the terminal; some will require participation of the host aplication. It is intended, however, that this division of labor be invisible to the user.

Hardware

During the first phase of the project an NLS work station was implemented with an HP2645, the ISI standard terminal. This implementation responded to requirements at ISI for additional NLS workstations and demonstrated our ability to take advantage of the processing power of the terminal. The implementation involved an I/O interface board, a mouse and keyset, and the integration of all the line processor functions of the NLS workstation into the HP2645 firmware. This implementation was repeated in the HP2648 graphics terminal to provide a graphics work station (although with limited screen size) for under $5,000 (cost of the traditional work station is approximately $12,000). These terminals are being used successfully at ISI and Gunter AFS, Montgomery, Alabama.

The second phase of development established design requirements for the MAST hardware, a set of functional capabilities to be implemented in terminal and host for the support of document preparation at ISI (1). These requirements are the basis for a set of working papers specifying the terminal hardware (2), an editor and formatter (3), and the firmware (4). The hardware specification led us to consider two approaches: one, a modification of the existing HP2645; the second, a new design based on an 8086 microprocessor, a bit map, and high-resolution screen. The first approach produced a prototype HP26XX-compatible terminal with a higher capacity screen (48 lines of 80 characters), which is currently being evaluated at ISI. We also wrote new firmware, based on our experience with the MME terminals, to satisfy most of the MAST requirements. The second approach was pursued until completing a paper design.

During the design process for the second approach we communicated with several terminal manufacturers who had working prototypes of terminals similar to the MAST model. Except for the PERQ System by Three Rivers Computer Corp., these prototype terminals were not scheduled to become products soon enough to be useful to the MAST project. Although the PERQ System is more capable, and considerably more expensive,

than is required for MAST, we determined that it is the least expensive and fastest way for ISI to acquire hardware for the MAST demonstration. As a result, we have terminated the firmware checkout of the prototype HP-based system and the development of new terminal hardware and have ordered a PERQ System for the MAST demonstration. We expect delivery of a PERQ in February or March of 1980.

Editor Application

A description of the editor is best understood in terms of our model of a document preparation system. The user of a documentation system should not have to communicate with the computer in any kind of language that is explicitly and visually embedded within the text he edits; he should modify the appearance of the document directly, rather than through commands to some later process, so that he can see the consequences immediately; at most, he might describe to the computer what kind of document feature he is using at any point with the expectation that doing so would cause both the screen and the final document to assume the proper appearance for the output device he has selected. In creating or modifying a document the user should work with a reasonably accurate representation of how the final product will appear; as he makes form or content changes, the display should automatically and immediately adjust itself to show the effect of these changes; what he sees while editing a document must be what he gets in the final hardcopy version.

The editor's database is a virtual document that has a rich enough description of the manuscript's appearance to drive output devices with a wide range of printing capabilities. The virtual document contains not only content but also indications of form. A user may modify both content and form, but he does so only in terms of the displayed manuscript; form indications in the virtual document, however, are independent of the output device (whether printer or display). To print or display a manuscript, the virtual document must be formatted: it must undergo an output transformation, based on the characteristics of the output device, in which the form indications are bound for that device.

Ultimately, the editor will be partitioned so that some of its functions run in the host and some in the terminal. The display formatter will run entirely in the terminal but will respond to queries from an application about the content and format of the display. A print-formatter interface module will have to be implemented in the host; it will use the

decisions made by the display formatter to create a symbolic file to be supplied to a printer formatter such as SCRIBE or a photocomposition system.

Initially, the display formatter and the terminal-resident part of the editor will run under a LISP interpreter in the terminal. An interpreter for a subset of INTERLISP-10 sufficient to support the editor and display formatter has been written in UCSD-PASCAL and is awaiting the terminal for checkout. (The PERQ System is basically a PASCAL system supporting the UCSD dialect; the inner CPU is a P-Machine.) An alternative approach is being explored that would compile the LISP application programs to a machine-independent form that can either be interpreted in the terminal by a more efficient interpreter or translated to executable code to run in the terminal.

Application Software Development

A display editor, suitable for use in a documentation system as described above, has been developed; it supports a full range of manuscript editing and formatting functions but not the final stages of document generation involving pagination, cross-referencing, front matter, and indexing. An incremental display formatter also has been developed to update display instructions as each editing operation takes effect. These MAST applications have been developed and checked out (to the extent possible without a suitable processing terminal) as INTERLISP-10 systems. This provides us with a rich, modern, dependable programming environment for writing and debugging the applications that will eventually be executed in a terminal. In examining the issues of downloading these programs (getting them to run conveniently in the terminal) we uncovered a series of interesting questions worthy of separate consideration: the issues relating to generalized solutions for split applications, downloading, partition boundaries, and cross-partition protocols are being considered at ISI in a separate project. We will of course provide one solution to these issues with implementation of the MAST demonstration applications.

An NVT application has also been written and partially debugged. An NVT is required to communicate with all current programs and is the default application for MAST. Consideration has been given to the NSW front end application, for which paper designs and working documents are available. We have not yet implemented or demonstrated these solutions. We will focus on these efforts after integrating existing software into the PERQ.

IMPACT

The major impact of the MAST effort will be noted after successfully integrating and demonstrating our existing software in PERQ. However, interim results are worth noting. The HP/NLS work station provides all users with an excellent alternative to acquiring the required hardware. It is economical, supported, resonably packaged, and compatible with all other software available to the user. At ISI it is particularly desirable since the HP26XX is our standard terminal and simplifies the in-house maintenance. ISI has already fabricated 25 terminals and plans 20 more in the near future.

The prototype large-screen terminal has also been well received by potential users in the ARPA community. It is compatible with all the existing software at ISI, including SIGMA and the MME firmware, and the higher character capacity adds a desirable dimension to the terminal/user interface.

The editor and terminal requirements documentation have had wide distribution in both the ARPA research and industry communities. MAST will provide a model for future terminals, a demonstration of a modern programming development system for terminal applications software, and a demonstration of a document preparation system for researchers.

FUTURE WORK

Our future effort depends on the acquisition of PERQ. We will evaluate PERQ as a terminal (decreasing its cost by eliminating unneeded capabilities), integrate the MAST editor into PERQ, and demonstrate the final product for ISI and outside evaluation. Additionally, we will evaluate PERQ as a personalized computer and integrate the system into the computer demand requirements at ISI.

REFERENCES

1. Cooperband, A. S., A documentation system, USC/Information Sciences Institute, WP-11, 1978.

2. Cooperband, A. S., R. Medina-Mora, L. Gallenson, Multi-application terminal requirements, 1978 (draft).

3. Cooperband, A. S., R. Medina-Mora, Preliminary display editor specifications, 1978 (draft).

4. Cooperband, A. S., R. Medina-Mora, P. Raveling, MAST system firmware design, 1978 (draft).

11. USER-DEDICATED RESOURCE

Project Staff: Chloe Holg

PROBLEM BEING SOLVED

The ARPA/IPTO research community's continuous expansion of the frontiers of information processing technology is reflected in a constant expansion and modification of information tools available on the ARPANET. These tools tend to have inadequate documentation; moreover, because of their large numbers and many subsequent modifications, it is difficult even to keep track of them all.

ARPA has been introducing new users to the network at an increasing rate for several reasons. First, the network is a superb communications medium. 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 via the ARPANET.

Before this project was established in 1977, the full benefits of network usage were not being fully realized for either ARPA or the military services. Users faced an initial barrier, partly because many were new to on-line computer systems in general, partly because system modification 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 an ARPANET Host.) The documentation that did exist, prepared by researchers or programmers for their own use, slighted the needs of nonprogrammer users.

GOALS AND APPROACH

The original goal of this project was to make a single individual responsible for helping new users overcome that initial barrier by keeping track of relevant developments and clearly communicating (i.e., with minimum computer jargon) about them. Since this user resource was introduced to the ARPANET community in 1977, its responsibility has expanded to include direct on-site tutorial assistance to military users at the military bases themselves, and involvement in the specification and implementation of full scale training programs for groups of military personnel taking part in long-range experiments

and tasks based on use of the ARPANET, e.g., the system orientation program for participants in the Army Data Distribution System (ADDS) experiment at Fort Bragg, NC.

PROGRESS

Although new users eventually become experienced users, there is no end to the problem; newer users are constantly introduced because of expansions of the IPTO program and the normal turnover in military personnel. These new users are provided with appropriate documentation as soon as their accounts are installed on the ISI machines, either directly from ISI or from the store of documentation provided to IPTO and other ARPANET sites.

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 the maintainers of system programs to better meet the users' needs.

With ongoing efforts among the system support groups to effect standardization of resources available in TENEX and TOPS-20 across the ARPANET machines, this task has been and is still providing broad support to users across the entire network. Military organizations provided with assistance during the lifetime of this project include the Advanced Command and Control Architecture Testbed groups, the System Development Laboratory and the Naval Ocean Systems Center, the Naval Research Laboratory, the Rome Air Development Center, the Naval Postgraduate School, the Army Signal School, the Strategic Air Command, and the Naval Intelligence Command.

IMPACT

This project has filled a long-standing need for simple, understandable, and above all, *new-user-oriented* (rather than system-oriented) introductory documentation and, in meeting this need, has been a large factor both in reducing the time required for a new user to become acclimated to the scope of the ARPANET's resources and in helping both new and old users make the transition to other operating systems with a minimum of confusion.

Frequently at sites supporting large groups of users, a point of contact between the site and this project has been established to channel problems and questions to the proper sources of information. Further, advice of changes to the operating systems are routinely provided; where applicable, updated quantities of documentation are made available for local distribution.

The first major task for this project was to produce *The Joy of TENEX* [1], a TENEX manual for novice users, since available TENEX manuals were simply unsuitable for that purpose. Both the rapid distribution of the manuals and the appearance of new documentation (e.g., *XED User's Manual: Beginning Instruction* [8]) necessitated a second printing 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 TIP connections for purposes of mail-handling, and a more basic, primer-like manual [2] was published in April 1977. The fourth version of *More Joy of TENEX* was completed, published, and distributed in 1977. The two ISI TENEX manuals were combined with TOPS-20 material to create a comprehensive two-volume set of basic reference manuals for new users [4, 5]; these were published and distributed in January 1979.

Also in January 1979, the System Orientation package was delivered to ADDS Experiment personnel [6]. This project made a major contribution to the development of the training techniques and strategies employed during the critical first phase of the ADDS Experiment during which approximately 400 new users were trained at Fort Bragg.

A preliminary general information Ada manual was prepared and delivered to DARPA in July 1979 [7].

The experimental USER program concept developed by this project and made available to a very limited set of users for purposes of evaluation was the basis for development of a name server program at another ARPANET site and presently is available on many of the ISI machines.

FUTURE WORK

This project will continue to provide assistance to both new and experienced users in the ARPANET community, in the areas of direct individual and group problem-solving and

comprehensive documentation. Revisions to the operating systems and subsystems are routinely monitored and documented. Work continues on new versions of both on-line and hardcopy user documentation. As time permits, work will proceed on the development of an on-line TENEX user general information system.

REFERENCES

1. Holg, Chloe, *The Joy of TENEX.;3 ...the basics* and *More Joy of TENEX.;4 ...some of the refinements*, USC/Information Sciences Institute, November 1976.

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

3. Holg, Chloe, *ARPA/NAVY/CINCPAC Military Message Experiment SIGMA Primer*, USC/Information Sciences Institute, TM-77-9, November 1977.

4. Holg, Chloe, *the Joy of TENEX and TOPS-20 ...in two parts, Volume I*, USC/Information Sciences Institute, TM-79-15, January 1979.

5. Holg, Chloe, *the Joy of TENEX and TOPS-20 ...in two parts, Volume II*, USC/Information Sciences Institute, TM-79-16, January 1979.

6. Holg, Chloe, ADDS Experiment System Orientation and Reference Manual.

7. Holg, Chloe, All about Ada, July 1979 (draft).

8. *XED User's Manual: Beginning Instruction*, USC/Information Sciences Institute, TM-76-3, May 1976.

12. ARPANET TENEX SERVICE

Technical Staff

Marion McKinley, Jr.
Serge Polevitzky
Wanda N. Canillas
Dale Chase
Phillip Crowe
Vernon Dieter
George Dietrich
Dwain Durden
Walt Edmison
Glen Gauthier
Kyoo C. Jo
James T. Koda
Wayne Kusaba
Kyle P. Lemons
Donald R. Lovelace
Raymond L. Mason
John P. Metzger
William H. Moore
Edward D. Mortenson
Robert Parker
Clarence Perkins
Vernon W. Reynolds
Dale S. Russell
Marilynne L. Sims
Barden E. Smith
Lee P. Taylor
Phyllis L. Taylor
Leo Yamanaka

Support Staff

Larry M. Akana
Larry Fye
Oralio E. Garza
James Griffin
Robert Hines
Richard E. Kaiser
Taylor W. Kidd
Archer J. Lewis
Robert Logsdon, Jr.
Keith Miles
Randolph Riedel
Robert M. Robbins
Ronald L. Ross
Gary Seaton
Ronald D. Shestokes
Audree Smith
Scot T. Smith
Zennithan Stringfellow
Patrick Wieber
Deborah C. Williams
Laura York

INTRODUCTION

The ISI ARPANET TENEX service project presently consists of three computer centers: one local and two remote. The former 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 4000 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 sites are operated in a classified environment. One installation is 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 involved in the joint ARPA and U.S. Navy Command and Control Experiment. This classified computer center is accessible by personnel physically located within the classified facility and via a secure ARPANET communication private line interface (PLI). This PLI communication technique allows the ACCAT system to service the Naval Post Graduate School (NPGS), Monterey Bay, California, and CINCPAC Fleet, Hawaii. On October 1, 1978, ISI assumed responsibility for a second remote installation. This center is operated in a classified environment as part of the Military Message Experiment (MME) at CINCPAC Headquarters, Camp Smith, Oahu, Hawaii.

The local computer center consists of five large-scale Digital Equipment Corporation (DEC) central processors (one KA-10, one KI-10, one KL-2060T and two KL1090T), Bolt Beranek and Newman (BBN) virtual memory paging boxes, large-capacity memories, on-line swapping and file storage, and associated peripherals (see Fig. 13.1). Three of the systems presently run under control of the TENEX operating system (originally developed by BBN) and two run under the DEC TOPS-20 operating system, both of which support a wide variety of simultaneous interactive users. In addition, the local facility supports other processors, such as several DEC PDP-11/40s, and one PDP-11/45, and associated peripheral devices.

The NOSC remote center consists of two large-scale DEC central processors, (one KL-2040T and one KA-10), a virtual memory paging box, large capacity memories, on-line swapping and file storage, and associated peripherals (see Fig. 13.2). The KA-10 runs under the TENEX operating system and the KL-2040T runs under the DEC TOPS-20 operating system.

The MME remote center consists of one large-scale DEC KL-1090 central processor, large capacity memories, on-line swapping and file storage, and associated peripherals (see Fig. 13.3). This system runs under a specially modified version of the TENEX operating system.

HARDWARE

In January 1979 ISI added its fifth large-scale central processor to the ARPANET, ISID. This system is a KL-2060T with one million words of internal MOS memory, five RP06 disk drives, one line printer, and associated peripherals. ISID is providing computer service to NLS users at the Air Force Data Systems Design Center (AFDSDC), Gunter Air Force Station, Montgomery, Alabama. This system also provides computerized message and editing capabilities to users from the ARPA-sponsored Army Data Distributions (ADDS) project at Fort Bragg, North Carolina. An additional 9-track tape drive was installed, bringing to three the number of tape drives available to ISID and ISIE. Two KL-1090Ts were acquired and one was used to upgrade ISIC from a KA-10 to a KL-10. The other KL-10 is being used as a software development machine. A CALCOMP 4-spindle disk system was acquired and installed onto the KL-10 development system. These disks will

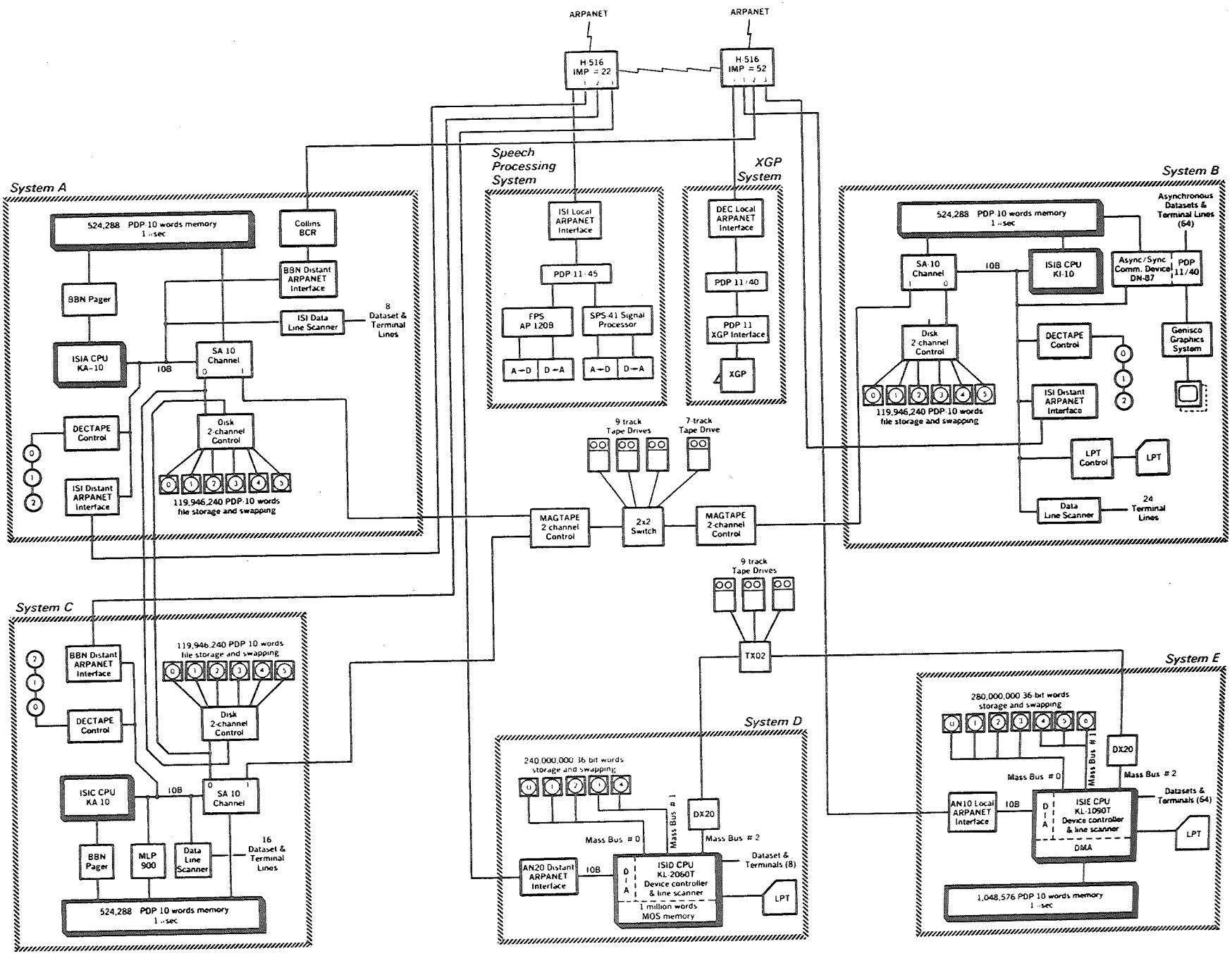


Figure 12.1. Diagram of local ISI ARPANET TENEX facility

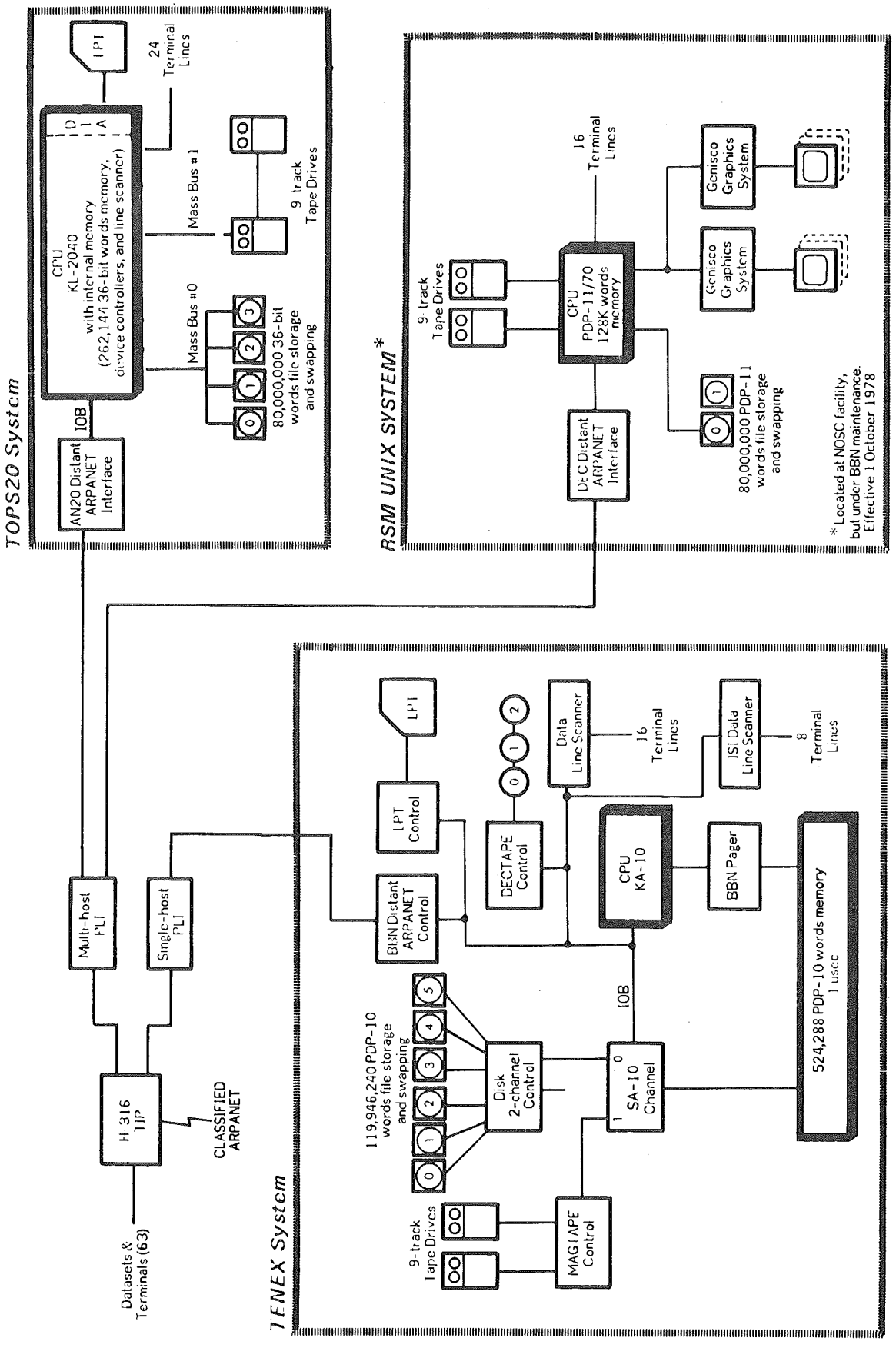


Figure 12.2. Diagram of remote ISI ARPANET TENEX facility at NOSC

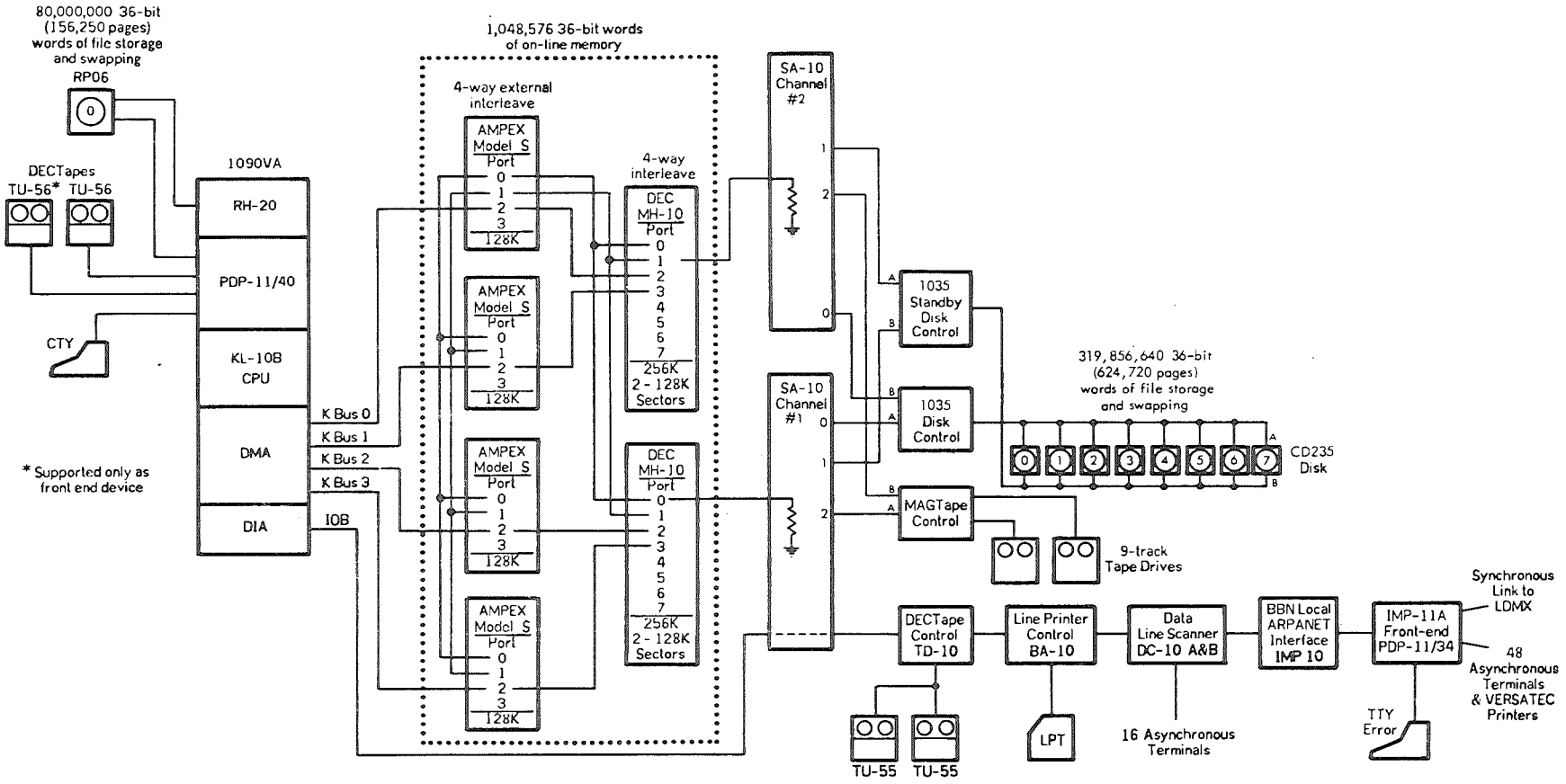


Figure 12.3. Diagram of remote ISI ARPANET TENEX facility at Camp Smith, Oahu

provide 312K pages of file storage and swapping space and allow either the TENEX or TOPS-20 operating system to be run. The ISI machine room was expanded by 600 square feet to accommodate the new KL-1090 development system.

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 Signal Processing System and a Floating Point Systems AP-120B (FPS) (configured as a speech processor), two microprogrammable processors, an MLP-900 and a QM-1, and several associated peripheral devices such as disk, memories, terminals from various manufacturers, and several special interfaces designed and developed by ISI. Each microprogrammable processor is slaved to one of the large-scale processors and is operated as if it were a subordinate process. These systems are used to emulate other hardware. The system based on the MLP-900, called PRIM, is operational; the system based on the QM-1, called QPRIM, is under development.

Local systems ISIA and ISIC are currently designated as priority systems and are therefore cross-connected (cabled) in such a manner that if one system crashes or is otherwise unavailable because of hardware/software maintenance or development, 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.

In October 1978 ISI assumed responsibility for the system operation of the Military Message Experiment located at Camp Smith, Oahu, Hawaii. In that same month ISI personnel removed two older KA-10 CPUs and installed a new KL-1090. This changeover was accomplished in one week and included not only the installation of new hardware but also the revamping of the computer room's a/c power and air-conditioning. This new system layout as configured included, besides the newer and faster KL-1090, one million words of core memory, two magtape drives, one line-printer, one RP06 disk drive, eight CALCOMP 230 disk spindles, two CALCOMP 1030 disk controllers, one SA10 channel, one data line scanner, and an ARPANET interface connected to a PDP-11/34 and the AUTODIN. Along with the change in hardware equipment, the operation and maintenance responsibilities were assumed by ISI by hiring a full staff of operators, hardware maintenance personnel, and system programmers. The ISI maintenance staff was also given the task of maintaining approximately 25 CRT Hewlett-Packard 2649 terminals and seven VERSATEC electrostatic printers distributed throughout offices of the Command

Center. In April 1979 it was decided to replace the CALCOMP 1030/230 disk system with a CALCOMP 1035/235 double density configuration. At this time another SA10 data channel was also installed to provide reliability and availability through redundant hardware accessibility to the tape and disk systems. The new disk system increased the file storage and swapping space from 300,000 to over 600,000 pages. A complete set of hardware spare parts was kept on site at Camp Smith to help minimize downtime. The site was maintained around the clock by ISI personnel. On October 1, 1979, at the conclusion of the MME, this system was removed from Camp Smith and shipped to ISI in Marina del Rey, California.

SOFTWARE

The prime concern of the software group is to provide stable software at a consistent level across all ISI TENEX and TOPS-20 systems. This includes the monitor, subsystem, and utility programs. The secondary concern is designing or introducing new programs or additions to existing programs to support various research needs.

The TENEX monitor used on all ISI TENEX systems is version 134 from BBN, with additions of a few utility JSYSs for local support, and minor bug fixes. Many of the bug fixes were obscure anomalies in file and process support JSYSs, which came to light during the development and testing of the SIGMA MME system. The major changes in TENEX this year were made to support the DEC KL-1090-VA system at CINCPAC Headquarters.

TENEX was modified to run on DEC's KL CPU, running in "KI mode," and was installed and operational at CINCPAC in October 1978. In April 1979, the ability to support multiple SA10s was added. The purpose was two-fold: to improve data throughput and to make maximal use of existing hardware. Improved throughput is realized when transfers may be spread between two (or more) SA10s, which may communicate with several controllers. All operational hardware may be kept on-line, and the software will determine at runtime what is available, and make the best use of what is found. An important side effect is the potential for faster recovery times from crashes due to failures in some components of this hardware. Those components that are duplicated may simply be switched off-line and the system rebooted without having to switch cables or diagnose and correct the failure at that time.

New disk/magtape code was installed on the local TENEX systems, as was new network code from BBN to support 96-bit network leaders and TCP 2.5. The local systems have only one SA10, so separating disk and magtape transfers did not increase throughput. The change did add a few bug fixes and the feature of checking for the presence of disk and magtape. The new network code is important because it allows communication with systems that may only be addressed with the new leader format. Also in this package was TCP 2.5, which allowed for testing the BCR encrypting device.

The TOPS-20 monitor used on all ISI TOPS-20 systems is version 3A from DEC, with local support additions and a few minor bug fixes. The change from version 101B to 3A was accomplished in May and July of 1979. The important changes made to 101B and carried into 3A included a pieslice scheduler, new global page management, increased disk storage and encrypted passwords. Also in July, new network code from BBN was added to support both 96-bit network leaders and TCP 4.0.

Subsystem maintenance is a continuous effort. The largest project in this area was conversion of subsystems to accommodate changes introduced by TOPS-20 version 3A. Most subsystems are still able to make system dependent selections at runtime, allowing the same binary files to be put on all ISI systems regardless of monitor type. The File Update Support System (FUSS) developed in 1977 continues to be a very effective tool for providing consistent levels of software across ISI's several systems.

Additional subsystem support was given to NLS 8.5, especially for the user groups at DARPA and AFDSDC. This included on-site training for AFDSDC personnel, operational assistance to users, bug fixes, and necessary modifications to accommodate changes introduced by TOPS-20 version 3A.

Work with the diagnostics was mainly concerned with insuring their operation on the KL CPU. The old diagnostics released by DEC for various peripherals continued to work, so we had to modify only locally developed and supported diagnostics, namely for the SA10 and IMP interface. There were two areas requiring attention, terminal I/O and paging. The SA10 diagnostic has also undergone improvements to allow testing multiple SA10 configured 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 NOSC 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. The ISI remote MME computer center is manned on a twenty-four-hour-a-day, seven-day-a-week basis. At least one operator is physically on-site at all times, and the systems programmers and computer service engineers are physically on-site or on a one-hour on-call schedule.

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.

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).

FUTURE WORK

As soon as TOPS-20 release 4 becomes stable and the computer center's modification of TOPS-20 to support nonstandard devices has been sufficiently tested, all systems will be converted to TOPS-20. The numerous benefits of TOPS-20 over TENEX include improved systems diagnostics (SYSERR, for example); better, more tolerant strategies with device errors; vendor support; address space in the EXEC and MONITOR, which allows support of more physical core; better supported subsystems (TOPS-20's TCP version 4 versus TENEX's TCP version 2.5, for example); and the possibility of

user-mode extended addressing under TOPS-20 (TENEX offers only 256K of user address space). Consolidating and standardizing under TOPS-20 will concentrate support efforts rather than divide them between two systems.

Following conversion to TOPS-20, the next effort will be to implement a shared-file system, under which computer systems will be paired. For example, ISIC might be paired with ISIE, ISIA with ISIB, and ISID with ISIF. Of the ISIC-ISIE pair, neither one would normally be aware of the other. Should the ISIC CPU fail, however, the disks constituting ISIC's file system could be switched to ISIE by setting a few hardware switches and invoking a privileged operating system command. Users who had been logged in to ISIC would then log in to ISIE until ISIC was repaired. Repair work on the ISIC CPU could proceed without blocking users of ISIC from their files. Note that if both ISIC and ISIE CPUs fail, no other systems could act as temporary processors for ISIC's or ISIE's file systems.

ISI PUBLICATIONS

Research Reports

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

Alfvin, Peter W., *A Formal Definition of AMDL*, ISI/RR-79-78, November 1979.

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.

---, Neil M. Goldman, and David Wile, *Informality in Program Specifications*, ISI/RR-77-59, April 1977.

---, Neil M. Goldman, and David Wile, *Meta-Evaluation as a Tool for Program Understanding*, ISI/RR-78-69, January 1978.

---, Neil M. Goldman, and David Wile, *On the Use of Programming Knowledge*, ISI/RR-77-63, October 1977.

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.

---, *Mathematical Approach to Computational Networks*, ISI/RR-78-73, November 1978.

Crocker, Stephen D., *State Deltas: A Formalism for Representing Segments of Computation*, ISI/RR-77-61, September 1977.

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.

Gerhart, Susan L., *Program Verification in the 1980s: Problems, Perspectives, and Opportunities*, ISI/RR-78-71, August 1978.

Goldman, Neil, Robert M. Balzer, and David Wile, *The Inference of Domain Structure from Informal Process Descriptions*, ISI/RR-77-64, October 1977.

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.

Guttag, John V., James H. Horning, Ralph L. London, *A Proof Rule for Euclid Procedures*, ISI/RR-77-60, May 1977; also in Neuhold, E. J., (ed.) *Formal Description of Programming Concepts*, North-Holland Publishing Co., 1978, pp. 211-220.

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.

Lesser, Victor, and Lee D. Erman, *An Experiment in Distributed Interpretation*, ISI/RR-79-76, July 1979.

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

---, and Neil M. Goldman, *Process Models of Reference in Context*, ISI/RR-78-72, October 1978.

---, and Armar A. Archbold, *Working Papers in Dialogue Modeling*, Volume I, ISI/RR-77-55, January 1977.

London, Ralph L., Mary Shaw, and William A. Wulf, *Abstraction and Verification in ALPHARD: A Symbol Table Example*, ISI/RR-76-51, December 1976.

Lynn, Donald S., *Interactive Compiler Proving Using Hoare Proof Rules*, ISI/RR-78-70, January 1978.

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

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

---, *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.

---, Greg Scragg, and Armar A. Archbold, *Working Papers in Dialogue Modeling*, Volume II, ISI/RR-77-56, January 1977.

---, *Dialogue Games*, ISI/RR-79-77, November 1979.

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.

Moriconi, Mark S., *A System for Incrementally Designing and Verifying Programs*, Volume I, ISI/RR-77-65, January 1978.

---, *A System for Incrementally Designing and Verifying Programs*, Appendix, Volume II, ISI/RR-77-66, January 1978.

Musser, David R., *A Proof Rule for Functions*, ISI/RR-77-62, October 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, August 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; also appeared in *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 4, December 1976, pp. 253-265.

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.

1977 Annual Technical Report: A Research Program in Computer Technology, July 1976-June 1977, ISI/SR-77-8, November 1977.

Carlstedt, Jim, *Protection Errors in Operating Systems: Serialization*, ISI/SR-77-9, April 1978.

Carlstedt, Jim, *Protection Errors in Operating Systems: A Selected Annotated Bibliography and Index to Terminology*, ISI/SR-78-10, January 1978.

Hayden, Charles, Peter W. Alfvin, and Stephen D. Crocker, *Multi-Microprocessor Emulation: Annual Report for 1977*, ISI/SR-78-12, April 1978.

Bisbey, Richard, and Dennis Hollingworth, *Protection Analysis: Final Report*, ISI/SR-78-13, July 1978.

1978 Annual Technical Report: A Research Program in Computer Science, July 1977-September 1978, ISI/SR-79-14, February 1979.

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.

Gallenson, Louis, Alvin Cooperband, and Joel Goldberg, *PRIM System: AN/UYK-20 User Guide/User Reference Manual*, ISI/TM-77-5, October 1977.

---, *PRIM System: U1050 User Guide/User Reference Manual*, ISI/TM-77-6, October 1977.

---, *PRIM System: Tool Builder's Manual/User Reference Manual*, ISI/TM-78-7, January 1978.

Holg, Chloe, *ARPA Navy CINCPAC Military Message Experiment SIGMA Primer*, ISI/TM-77-9, December 1977.

Oestreicher, Donald R., Paul Raveling, and Robert H. Stotz, *HP/MME Terminal Application Specification*, ISI/TM-78-10, March 1978.

Rothenberg, Jeff, *DARPA Navy CINCPAC Military Message Experiment: SIGMA Message Service Reference Manual*, ISI/TM-78-11, March 1978.

Rothenberg, Jeff, *DARPA Navy CINCPAC Military Message Experiment: SIGMA Message Service Reference Manual*, ISI/TM-78-11.2, June 1979.

Holg, Chloe, *the JOY of TENEX and TOPS-20...in two parts: Part One* ISI/TM-79-15, March 1979.

Holg, Chloe, *the JOY of TENEX and TOPS-20...in two parts: Part Two* ISI/TM-79-16, March 1979.

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