ISI/SR-81-19

1980
Annual Technical Report
VOL. 1

October 1979 - September 1980

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

1980 Annual Technical Report

A Research Program in Computer Technology

ISI/SR-81-19

Unclassified

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

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ISI/SR-81-19	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* 1980 Annual Technical Report: A Research Program in Computer Technology		5. TYPE OF REPORT & PERIOD COVERED Annual Technical Report October 1979 - September 1980 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 ARPA Order # 2223
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE August 1981 13. NUMBER OF PAGES 145
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 programming, domain-independent interactive system, HEARSAY III, natural language, nonprocedural language, nonprofessional computer users, problem solving, problem specification, process information
2. abstract data type, abstraction and representation, Affirm, Alphard, Euclid, interactive theorem

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, 1979, to September 30, 1980, for the Defense Advanced Research Projects Agency. The research applies computer science and technology to areas of high DoD/military impact.

The ISI program consists of fourteen research areas: Specification Acquisition From Experts - study of acquiring and using program knowledge for making informal program specifications more precise; Program Verification - logical proof of program validity; Autopsy - research on source-to-source program translation combining automatic techniques with an interactive system to provide the

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

Unclassified

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

19. KEY WORDS (continued)

proving, Pascal, program correctness, program verification, rewrite rules, software specification, verification condition

3. Ada, Autopsy, CMS-2, DOD-1, program conversion, program equivalence, program translation, source-to-source transformation
4. Ada, denotational semantics, formal definition, formal semantic definition, typed lambda calculus
5. implementation of interactive systems, knowledge base, knowledge-based inference, natural interface, online services, process script, service building, tool building, user interface
6. command and control graphics, computer graphics, high-level graphics language, on-line map display
7. computer mail, gateways, interconnection, internetwork protocol, networks, protocol design, protocols, protocol verification, transmission control protocol, type-of-service
8. computer network, digital voice communication, network conferencing, packet-switched networks, secure voice transmission, signal processing, speech processing, vocoding
9. communication protocols, cooperation between decentralized processes, DSN modeling, packet radio network, position-location, rigidity, sensor networks
10. command and control, digital voice communication, graphic input device for terminal, multimedia communications, portable terminal, radio-coupled terminal
11. application splitting, central processor, dedicated local processing, distributed processor system, man-machine interface, partitioning, resource allocation, responsiveness, terminal processor
12. document preparation, editor/formatter, Network Virtual Terminal, partitioning, state-of-the-art terminal, virtual document
13. AN/UYK-20, emulators, ISPS, microprogramming, MULTI, multimicroprocessor emulation, National Software Works, program development tools, QM-1, QPRIM, Smite
14. application software, ARPANET, customer service, hardware, interface, computer network, KA/KI, KL10/KL20, operations, PDP-11/45, resource allocation, system software, TENEX, timesharing, TOPS-20

20. ABSTRACT (continued)

human manager complete control over the translation process; Formal Semantics - development of tools and methodologies to support the formulation of precise, readable, and accurate formal semantic definitions; Cooperative Interactive Systems - construction of a system to provide natural input/output and help facilities for users of interactive services; Command and Control Graphics - development of a device-independent graphics system and graphics-oriented command and control applications programs; 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; Network Secure Communication/Wideband Communication - development of protocols and real-time systems to transmit digitized voice over the ARPANET and development of technology required for the future support of thousands of simultaneous conversations being transmitted over a wideband satellite channel in the internetwork environment; Distributed Sensor Networks - formulation of algorithms and communication protocols to support the operation of geographically distributed sensors; Personal Communicator - 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; Application Downloading - research on downloading interactive applications into a terminal, resulting in a decrease in demand for shared central processor time; 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; QPRIM - production of an online interactive emulation facility housed in an existing mature operating system; and Computer Research Support - operation of TENEX and TOPS-20 service and continuing development of advanced support equipment.

ISI/SR-81-19

1980
Annual Technical Report
VOL. 1

October 1979 - September 1980

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 Executive 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
Tardia Dinkins
Gail Geppert
Judy Gustafson
Maureen Jester
Toni Leon
Gina Maschmeier

Graphics Manager:
G. Nelson Lucas

Librarian:
Sally Hambridge

Publications:
Jim Melancon

Secretaries to Directors:
Patricia A. Craig
Joyce K. Reynolds

CONTENTS

SUMMARY

This report summarizes the research performed by USC/Information Sciences Institute from October 1, 1979, to September 30, 1980, for the Defense Advanced Research Projects Agency. The research applies computer science and technology to areas of high DoD/military impact.

The ISI program consists of fourteen research areas: *Specification Acquisition From Experts* - study of acquiring and using program knowledge for making informal program specifications more precise; *Program Verification* - logical proof of program validity; *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; *Formal Semantics* - development of tools and methodologies to support the formulation of precise, readable, and accurate formal semantic definitions; *Cooperative Interactive Systems* - construction of a system to provide natural input/output and help facilities for users of interactive services; *Command and Control Graphics* - development of a device-independent graphics system and graphics-oriented command and control applications programs; *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; *Network Secure Communication/Wideband Communication* - development of protocols and real-time systems to transmit digitized voice over the ARPANET and development of technology required for the future support of thousands of simultaneous conversations being transmitted over a wideband satellite channel in the internetwork environment; *Distributed Sensor Networks* - formulation of algorithms and communication protocols to support the operation of geographically distributed sensors; *Personal Communicator* - 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; *Application Downloading* - research on downloading interactive applications into a terminal, resulting in a decrease in demand for shared central processor time; *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; *QPRIM* - production of an online interactive emulation facility housed in an existing mature operating system; and *Computer Research Support* - operation of TENEX and TOPS-20 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 Specification Acquisition, Program Verification, and Autopsy projects should be considered as individual parts of an overall research effort in programming methodology and quality software; Wideband Communication, Distributed Sensor Networks, Internetwork Concepts Research, Command and Control Graphics, Personal Communicator, Cooperative Interactive Systems, Application Downloading, and MAST are linked elements of a major investigation into man-machine and network communications technology. This mutual reinforcement among the various projects at ISI contributes largely to the productivity of the Institute's research activities.

Specification Acquisition From Experts. This work is directed at helping people create unambiguous, consistent, and complete formal program specifications through informal description. While end users are quite capable of providing informal process-oriented descriptions of the task being automated, formalisms of any kind provide major impediments. The informal descriptions are characterized by partial, rather than complete, constructs. The system uses a knowledge base of program well-formedness rules to disambiguate and complete the informal natural language descriptions. An early version of the system has successfully converted several small informal specifications into formal specifications. Attention is now focused on handling large specifications through incremental formalization; to support this, a generalized AI architecture for knowledge-based systems, based on the Hearsay-II blackboard model, has been designed and implemented.

Program Verification. In its most restricted sense, program verification is the task of proving the consistency of a given program with its given specifications (what is often called the "correctness" of the program). The purpose of verification may be either (positively) to establish consistency for certification purposes or (negatively) to discover inconsistency as a step in debugging. In a broader sense, program verification is an analytic and synthetic tool for addressing a wider range of software quality considerations than just the consistency of the final specification and final program. The problem being solved by this project is two-fold: (1) the specific problem of finding and refining the appropriate mathematical proof methods and (2) the more general problem of integrating these methods with the software design and development processes. In 1980, the project experimented extensively with a usable verification system, ***Affirm***, supported outside users, and applied the system to a major effort in communication protocols.

Autopsy. The Autopsy project has been investigating translation of programs from old languages into new languages. While it is clear from both theory and practice that translation between *arbitrary* languages is not likely to succeed, the less ambitious goal of translation between "compatible"

languages may be quite reasonable. The approach has been to develop an interactive system that contains an array of *tools* for the user to deploy in translation. These tools must exist within a coherent framework, so an interactive monitor to govern access and use of these tools has been designed. Central to the design of the tools is the Autopsy intermediate language, which is used for representing, editing, and verifying the translations. The work of this project has focused on translation between a specific pair of languages, CMS-2M and Ada.

Formal Semantics. The principal goals of this project are the development of tools and methodologies for supporting the development of precise, readable, and accurate formal semantic definitions of programming languages. The specific research focus of the project is building tools for manipulating, processing, implementing, and testing the formal definition of Ada written by a group at the Institut National de Recherche en Informatique et en Automatique in France.

Cooperative Interactive Systems. The Consul system is designed to provide a natural, consistent interface for the user services of an interactive environment. User activities such as sending messages, maintaining an appointment calendar, generating graphical displays, now handled by separate subsystems, will all be accessible through a single interface that allows natural language input and provides help to the user. This is achieved through knowledge-based inference on a detailed model of user and system behavior. Research issues include knowledge representation and inference techniques, acquisition of domain-dependent knowledge, explanation, and software methodology. The current prototype system demonstrates natural interaction with a message service.

Command and Control Graphics. As more command and control information is maintained in computer form, computers will need to take a more active role in the presentation of that information. To facilitate the decision-making process, online computer-generated color graphics will replace binders of batch-generated printer listings. The purpose of this project has been to develop a system architecture to meet current and future C2 graphic requirements, with particular attention paid to adaptability to available computation, communication, and display resources, usability in a transnetwork environment, and ability to support the creation of pictures for use outside the immediate application environment. The work has resulted in the definition and implementation of a C2 Graphics System on the ARPANET. Future work will include the transfer of the graphics system to small, powerful 16-bit microprocessors and the development of a briefing-aid application.

Internetwork Concepts Research. This project explores the design and analysis of computer-to-computer communication protocols in multinetwork systems. The project has three task areas: (1) Analysis, (2) Applications, and (3) Design and 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 and Concepts is concerned with the development of network and transport protocols, in particular the Internet Protocol and TCP, and seeks new approaches in the application of packet switching to communication problems.

Network Secure Communication/Wideband Communication. The ISI Network Secure Communication (NSC) Project has been instrumental in the development of protocols and real-time systems to transmit digitized voice over the ARPANET, both in point-to-point conversations and multisite conferences. The project is now broadening its scope as the Wideband Communication (WBC) Project, which will develop the technology required for the future support of thousands of simultaneous conversations being transmitted over a wideband satellite channel in the internetwork

environment. It will advance packet voice from a demonstration program to an experimental system continuously available for use in the transaction of normal daily business. While the NSC project concentrated on voice communication, the WBC Project will work on integrated communication of several media, including voice. The goal is to develop real-time multimedia teleconferencing using wideband packet-switched networks. The initial emphasis other than voice will be on the development of a video bandwidth compression system which operates in real time and takes advantage of the ability of a packet-switched network to accommodate varying bandwidth requirements.

Distributed Sensor Networks. The major goal of the DSN project was to develop and export technology aiding the implementors of distributed and decentralized systems. The first step in this direction was derivation of an adequate description of the DSN problem space. This included 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, the goal was to develop objective methods of evaluating a system's performance. The research was directed toward the general problems of DSNs, particularly, system issues that were both sensor- and scenario-independent. The following areas were studied: the communication problem, the architecture problem, the organization problem, the user interface problem, the position-location problem, and the distributed algorithm problem.

Personal Communicator. This effort explores the design and application of personal, portable computer communication terminals that provide text, graphic, and voice modes and synchronized combinations of them. The existence of full-time, location-independent, handheld access to communications will allow fuller exploitation by the services of the area coverage and dynamic flexibility of packet radio. The goal of the exploratory design and experimental operation of model communicators within a multimode message system is to expose the functional and architectural parameters of the terminal and its user interface behavior, for the guidance and support of future designs of operational portable communicators and their associated systems.

Application Downloading. The rapid advancement in the processing capabilities of computer terminals provides an opportunity to allow terminal processors to share in the execution of application programs. The downloading of interactive applications into a terminal may often result in a decrease in demand for shared, central processor time. However, developing such distributed application programs is a difficult task. This difficulty is mitigated by first allowing an interactive application to be designed, implemented, and tested using standard development techniques in a single-processor environment; and then semiautomatically partitioning and optimizing the application in a distributed processor system consisting of a terminal and a host computer.

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.

QM-1 Programming Research Instrument. The QPRIM effort is producing an online interactive emulation facility housed in an existing mature operating system. QPRIM aims to bring program execution and testing into the programmer's working environment without paying the typically prohibitive cost involved in utilizing a simulation program on the development host. The PRIM project built such a prototype facility within the TENEX operating system; that facility was operational at ISI

from 1974 until 1979. QPRIM is a PRIM-like emulation facility running under the DEC TOPS-20 operating system and utilizing a production-emulation engine, the Nanodata QM-1. An interface unit between the DECsystem 20 and the QM-1 was installed in March 1979. The facility has been running and available at ISIB since June 1980. As part of the system shakedown, an emulator for the AN/UYK-20 will be written and tested.

Computer Research Support. ISI supports, operates, and maintains one TENEX and five TOPS-20 systems at ISI on a schedule of 164 hours per week. TENEX/TOPS-20 service is provided both to ARPA and to its research projects via the facilities at ISI. ISI also operates one TENEX and one TOPS-20 system at a computer center that is part of the Command and Control Testbed at the Naval Ocean Systems Center, San Diego, Calif. The Institute provides support for the Penguin at ARPA-IPTO, NLS user support, and NLS software support. The Institute also provides remote training and documentation for its military users in order to allow them to make the most effective use of the available facilities.

1. SPECIFICATION ACQUISITION FROM EXPERTS--SAFE

Research Staff:
Robert Balzer
Jeff Barnett
Don Cohen
Lee Erman
Neil Goldman
Philip London
David Wile

Support Staff:
Joan Elliott

1.1 PROBLEM BEING SOLVED

With the increasing sophistication and complexity of weapon and logistical systems, and the decreasing cost of computer hardware, the role of software within the military is becoming increasingly critical. System capabilities and costs are increasingly being determined by the complexity, costs, and lead times of the embedded software. The demand for such software far exceeds the military's capability to produce it.

Inadequate (informal) specifications have repeatedly been identified as the prime cause of poor software.[1] Much attention has been focused on the creation and use of formal specifications to help alleviate this problem. Unfortunately, writing such specifications is quite difficult because of the formalism itself, the need for absolute consistency, and the need for massive amounts of detail.

For this reason, we have been developing a tool which aids users by converting their informal specifications into formal ones. The user benefits of such informal specifications are that they are familiar (there are even MIL-SPECS describing them), they suppress information that is (thought to be) easily inferable, and they are far more understandable. Ultimately, our tool will allow users to continue enjoying these benefits as well as those arising from the existence of a formal specification: its use as an implementation contract, as a testable system prototype, and as an analyzable object to determine its properties and/or deficiencies.

In addition to our central goal of developing a theory of understanding and formalizing informal specifications, we have developed two important by-products which have influenced, and are being used by, other ISI projects: a formal specification language (Gist) and an AI expert-writing system (Hearsay-III).

---

[1] CCIP-85 study, High Cost of Software Conference.

1.2 PROGRESS

1.2.1 SAFE

The prototype SAFE system has understood and correctly formalized several real-world, albeit simplified, specifications extracted from actual MIL-SPEC 490-B5 specification manuals. It then successfully handled twenty-five perturbations of these examples as a demonstration of its robustness on such small simplified specifications. We believe these results demonstrate the basic feasibility of our approach.

The remaining watershed issue for our line of research is whether this approach can be scaled up to handle large, practical-sized specifications. Recognizing the inherent limitations of the prototype system (depth-first backtracking as a search paradigm, a three-pass "compiler" organization, procedurally embedded knowledge, and an inadequate formalism for expressing specifications), we completely rebuilt the system. This proved most difficult because, in addition to scaling the system to handle specifications that are almost two orders of magnitude larger, it was necessary to simultaneously change both the form of the knowledge used and the basic paradigm of the system.

These two changes were essential not only for the current work (applying the system to large, unsimplified real-world specifications) but also for the work we are planning. The first change (knowledge format) enabled us to substitute a rule-based expert system for the ad hoc, procedurally embedded knowledge in the old system. This new format provides a much more adaptable knowledge base to augment as we explore further the flexibilities of informal specification. This change has been completed and is based on the Hearsay-III system (see section 1.2.3).

The second change (the paradigm employed) provides a much sounder methodological foundation for further work. Unlike the old system, which processed the entire input through three successive passes, the new system incorporates a single new sentence into the current interpretation(s). As a result, the grain size of an experiment has been drastically changed from handling an entire specification to extending an interpretation to include more information (one sentence worth). As we are basically building a theory of how informal specifications can be understood and formalized, running experiments is crucial to the iterative development of that theory. Reducing the grain size to a single sentence makes it far easier to run such experiments. This change has been completely designed and partially implemented.

1.2.2 Formal Specifications

We developed an adequate formal language (Gist) to express specifications. This language differs from other specification languages in its ability to formally express constructs commonly found in informal specifications (rather than relying on provability criteria), its database viewpoint, its commitment to operationality, its tolerance of incompleteness, and its avoidance of implementation issues. This ARPA-developed language is the basis for a specification validation project sponsored by RADC, a transformation-based development system sponsored by NSF, and a new ARPA project that is developing mappings from these specification constructs into implementations. In addition to these roles, it also is the output language for SAFE and is beginning to act as its domain of expertise (see section 1.4.1). Gist thus occupies a central position in our effort.

Among the unique features of this language are:

1. Semantic use of constraints. Rather than treating constraints as redundant specifications which could be used for checking consistency, Gist uses them to restrict the acceptable interpretations for nondeterministic constructs. Only those choices which don't violate constraints now, or in the future, are acceptable. This use of constraints provides an extremely powerful formal specification construct which enables us to describe a process behaviorally, rather than algorithmically, in a way closely paralleling the informal way we resolve ambiguity in natural language.
2. Database view including inference. Gist represents the current state of a modeled system as a global database of relations among typed objects. Actions cause these relationships to change. The state of the database and of the objects within it is accessed through a uniform information extraction (query) language. This language locates objects through descriptions of their relationships to other objects. Furthermore, it hides the distinction between explicit and implicit information by including inference within the information-extraction language. Since a major portion of any computing system is concerned with spreading the effects of changes that occur, the self-organizational aspect of the automatic inference mechanism allows this portion of the system to be completely suppressed from the specification.
3. Historical reference. In addition to providing a uniform method of extracting information from the current state, Gist provides the same capability for all previous states of the system. This means that when historical information is needed, it can be directly referenced in the specification. This direct reference avoids having the specifier invent and maintain an auxiliary data structure, and closely approximates the informal mechanisms used in natural language.

1.2.3 Hearsay-III

We designed and implemented an AI expert-writing system called Hearsay-III, as a base for the revised SAFE system. It is currently being used by SAFE and two other ISI projects. Hearsay-III is a domain-independent tool for constructing knowledge-based systems. Its target is large, complex problems, for which many interacting sources of knowledge are required, specifically including those whose solutions are synthesized incrementally from partial solutions, and excluding "diagnosis" problems in which a solution is chosen from among an easily enumerated set.

The Hearsay-III system was designed to provide facilities to represent, compare, and pursue competing, incrementally constructed problem solutions. It is intended to support experimentation with long-term, large-system development. In particular, the design goals of the Hearsay-III system focus on development and debugging of theories of expertise in an application domain; its main goal is to allow for study of a domain, rather than to construct an expert performance system.

Hearsay-III adopts the heritage established by the Hearsay-II speech understanding system in that many of the mechanisms available to the system builder in Hearsay-III are generalizations of mechanisms present in Hearsay-II. However, Hearsay-III is domain-independent; it has no specific knowledge of speech understanding or any other domain at the outset.

Hearsay-III provides primitives that encourage a certain architectural style in the target system. This architecture includes the following features:

1. A structured workspace (called a blackboard) encouraging a hierarchical representation of data organized by various levels of abstraction.
2. Condition/action rules (called knowledge sources) which react to situations represented

on the blackboard, and whose action components can modify the blackboard. Condition satisfaction and action invocation are independently controllable events. The action component of a knowledge source is arbitrary Interlisp code, and thus all of its mechanisms for composition and abstraction can be used to raise the level of action expression above that of the Hearsay-III system primitives.

3. A metalevel problem solver to which the same facilities are available (i.e., it can use a scheduling blackboard and scheduling knowledge sources). The task of this problem solver (called the scheduler) is to resolve conflicts among the potentially large number of knowledge sources whose conditions have been satisfied. The scheduler chooses the next knowledge-source action to execute. The ability to construct a knowledge-based scheduler that reasons about pending knowledge-source activations provides an important step in the direction of separating competence knowledge from performance knowledge in the target system.
4. Facilities for explicitly representing decisions as data. Choices being considered by a Hearsay-III problem solver can be encoded on the blackboard. Decisions represented in this way can satisfy knowledge-source patterns, and can be modified by knowledge-source actions. Decisions therefore can be reasoned about in much the same way as other objects appearing on the blackboard.
5. The blackboard and all publicly accessible Hearsay-III data structures are represented in a relational database (called the AP3 database). The AP3 database is similar to those available in languages such as PLANNER, but it also includes strong typing for each of the relational arguments in both assertion and retrieval. These typed relational capabilities are available for directly modeling the application domain.
6. A context mechanism, which allows independent pursuit of competing approaches to solving a problem. The context mechanism allows several different versions of the blackboard, representing competing partial problem solutions considered in separate worlds, to be examined and manipulated independently by the knowledge sources.

1.3 RESEARCH ISSUES

1.3.1 SAFE

The SAFE system is a knowledge-driven system based on four types of knowledge: detecting informal constructs, proposing alternative interpretations, testing interpretations, and augmenting interpretations (to make them acceptable). Each of the knowledge bases is incomplete and ad hoc, having been built on an as-needed basis from our test specifications. With the system rebuilt in Hearsay-III, it is now possible for us to easily add new knowledge to each of these areas in a systematic manner.

In addition to using these four types of knowledge, which deal directly with the task of formalizing informal specifications (called the competence knowledge), the SAFE system must also use another type of knowledge to decide which alternative interpretations of the informal constructs to explore and incorporate into the formalized specification. One of the big advantages of the use of Hearsay-III is that this knowledge (called the performance knowledge), which guides the search through a space of possibilities by controlling and scheduling the competence knowledge, is also represented as a knowledge base of rules. Thus, just as the competence knowledge is accessible and augmentable, so too is the performance knowledge.

As part of our current effort, we are slowly building up the performance knowledge. This process is slow because we, and the research community in general, have little experience with such metalevel performance knowledge and because, as opposed to the competence knowledge which has existed

(albeit in procedurally embedded form) throughout this effort, no performance knowledge previously existed.

The old system simply used depth-first backtracking guided by well-formedness constraint violations to explore the space of possible interpretations. It is well known that this technique does not scale up, and it was abandoned as we began to explore large specifications. Our utilization of performance knowledge for SAFE is thus quite recent.

1.3.2 Manipulation and Readability of Formal Specifications

Through the development of the Gist formal specification language during our current IPTO effort, and as part of our RADC effort to apply Gist to military specifications, we have acquired some experience with formal software specifications. We have discovered that while the Gist language has made considerable progress towards simplifying the creation of adequate formal specifications, mainly through the availability of appropriate semantic and modeling constructs, it has not helped the specification readability problem.

This lack of readability of formal specifications (for both Gist and other formal specification languages) is a major impediment to the utility of formal specifications. It arises from a simplistic mathematical orientation towards formal specifications which (overstated) holds that all the necessary information merely needs to be stated (and be self-consistent) in the formalism together with the rules of inference. Then any derived information can be mechanically computed.

This view unfortunately neglects to recognize that, for people, form is as important, if not more important, than content. Thus, in addition to ensuring that all needed information is stated and is self consistent, the way that it is structured and presented is crucial for understanding. The lack of readability of formal specifications is a direct result of this oversight. None of the structuring and presentation mechanisms which have proved so valuable in informal descriptions have been adopted for formal specifications.

1.3.3 Hearsay-III Scheduling

The Hearsay-III system has been developed as a knowledge-based expert-writing system. It provides a general framework for separating domain-specific competence knowledge (methods) from performance knowledge (scheduling). The full power of the Hearsay-III system, in the form of a structured blackboard and cooperating knowledge sources, is available for implementing scheduling policies.

The state of the art with respect to such scheduling policies is quite primitive, so much experimentation is required. For this reason, we designed Hearsay-III with extremely general scheduling primitives, and with the ability to build and utilize arbitrary data structures to control scheduling policies. In fact, we believe that advances in scheduling expertise will be directly correlated with the sophistication of these data structures coordinating the individual pieces of scheduling knowledge, rather than with the addition of any particular knowledge source.

Furthermore, we have noticed a serious deficiency in the control structure of current expert systems for which Hearsay-III is ideally suited as a research tool. These systems are not able to alter

their mode of interaction in response to the capabilities of the user and/or the demands of the particular task. Rather, the respective responsibilities and roles of the system and the user are predetermined during the system design. This rigidity arises from the lack of separation of control from competence within the expert system.

Finally, we have observed that current expert systems are extremely limited in their ability to use "measures" applied to knowledge. The existing technology, as evidenced in Mycin and SRI's inference nets, allows one-dimensional numerical credibilities to be combined. However, knowledge can be (and is) quantized in many other ways in addition to its credibility, such as its utility, importance, costs, and timeliness. Each of these quantized measures should affect the way in which the knowledge is used. Such a multidimensional reasoning mechanism does not yet exist.

1.4 FUTURE WORK

Our research goals for the next few years are divided into three areas: one related directly to the SAFE system, and one each to the Gist and Hearsay-III by-products.

1.4.1 SAFE

By providing a well-defined space of allowable specifications, the design of the Gist formal specification language was the first step in codifying the four knowledge bases within the SAFE system. The next step is to define in terms of our internal input language the allowable types of informality (such as omitted operands and operations, ambiguous references, and unsequenced operations). With these classes of informality clearly defined, we can identify and incorporate into the system the appropriate additional knowledge that the four SAFE knowledge bases need to process the full range of these informal constructs.

To prevent the slow buildup of performance knowledge from limiting our progress on handling large specifications, we will adopt a policy of using manual intervention to guide the search appropriately. Gradually the emerging performance knowledge will replace this manual intervention. It should be noted that our perception is that this early reliance on manual scheduling is not an aberration, but rather a general paradigm of expert system creation based on the difficulty of explicating the performance knowledge and its dependence upon the competence knowledge being controlled.

In addition to these improvements in the system's robustness and predictability through systematizing its competence and performance knowledge bases, its ability to understand an informal specification can be enhanced by providing an extensive predefined domain model. This domain model is central to the system's operation. It provides the domain-specific knowledge which augments the built-in domain-independent knowledge to disambiguate the informal constructs. It contains the type structure of the domain, the possible relationships between objects and restrictions on those relationships, the actions of the domain including their operands and results, and the rules of inference. In short, it contains all the declarative information necessary to describe a domain in which the behavior defined by the specification is to occur.

Thus, the existence of an extensive, accurate domain model as a precursor to formalizing a specification provides a strong semantic support for the disambiguation of that specification and

suggests a mode of operation which capitalizes on this semantic support to increase the system's utility. Naturally, this model can itself be built by SAFE from informal input through its ability to dynamically acquire a domain model; but in this mode of operation the formalization of the domain model will be completed before the formalization of the specification itself begins.

The pre-existence of the domain model will provide a much stronger semantic base for the disambiguation of the specification and will remove any reliance on the capability to dynamically change the domain model--one of the most difficult problems for SAFE (this is basically the incremental compilation problem in which declarations change).

We will also require the pre-existence of the linguistic model, which will permit mapping from the vocabulary and syntax of the natural language front end to the names used in the predefined formal domain model.

The SAFE system currently makes the basic assumption that the specification being described is correct, i.e., there exists at least one valid interpretation (formalized specification). It has no mechanisms to deal with specifications that are really wrong (either because the user has not foreseen some interaction, or because the design is incomplete). A small step has been taken in this direction as part of our current effort to process specifications incrementally. We have given the system the capability to delay forming an interpretation of some construct until further information is provided. But this capability merely delays the decision making and doesn't address the issue of having intended interpretations which are not well formed. The current strategy of rejecting such interpretations must be softened so that analysis can proceed even when a specification violation is detected. In particular, such a violation should not preclude the disambiguation of other portions of the informal specification. This will be accomplished by annotating the interpretations with any well-formedness rules they violate and using these annotations to form a relative evaluation of the acceptability of alternative interpretations.

1.4.2 Manipulation and Readability of Formal Specifications

We would like to produce language forms and computer-based tools to enhance the readability and understandability of formal specifications. Specifically, we plan to identify the understandability mechanisms commonly used for partitioning informal descriptions (such as overviews and summaries, elaborations and refinements, normal case descriptions, and alternative viewpoints). We will also provide reading tools which utilize electronic windowing to focus on individual pieces of the specification and navigate between them in a "story unfolding" manner. Within such a framework, we will develop language forms and good practices for creating specifications through an elaboration/refinement paradigm.

We would also like to study the use of both static and dynamic graphics to provide alternative "redundant" viewpoints. We are particularly interested in developing the capability of restructuring multiple-participant systems from a participant-oriented (control) viewpoint to an object-oriented (data flow) viewpoint, and vice versa.

As part of our RADC effort, we will produce a tool for explicating implicit interactions among parts of the formal specification. We would like to incorporate this capability into our specification-reading tool.

Finally, the Gist language design is incomplete. It focuses on the logical, behavioral description of systems. Two major areas, error handling and interface specification, have yet to be defined. These facilities are common in programming languages but have not been adequately explored in the context of formal specification languages. The appropriate representation- and implementation-independent abstractions need to be identified as part of this design effort.

It is clear that both these issues complicate the system specification, making it more difficult to understand. It is therefore important to design these language facilities within the context of the elaboration/refinement paradigm and the specification-reading tool.

1.4.3 Hearsay-III Scheduling

We believe that Hearsay-III provides a particularly good framework for exploring scheduling policies and mechanisms, and we would like to pursue this basic scientific question within the context of the SAFE system and expert systems in general. In particular, we would like to identify, codify, and study the scheduling policies being discussed today (such as subgoaling, focus of attention, agendas, and resource-limited scheduling) and their use of data structures in the context of a fixed set of competence knowledge in the SAFE system to determine (as metaknowledge) the appropriateness, or lack thereof, of these policies and how they could usefully interact.

This will be accomplished by building generic versions of these scheduling policies, applying them to portions of the SAFE system, and determining the advantages and disadvantages of each. Then we will attempt to develop and test a technology for combining these policies to enhance the overall performance of the SAFE system.

Hearsay-III is an ideal framework for studying the problem of dynamically shifting responsibilities between a user and the system because it both separates aspects of the system and provides the data-structuring and dynamic-knowledge-source-coordination mechanisms needed for necessary run-time context-sensitive control decisions.

This issue will be addressed by devising a scheduler which, given a split of responsibilities between a user and the system, structures a fixed set of competence knowledge to accomplish the system's responsibilities in its interactions with the user. Such a scheduler will use metaknowledge to determine the best way to employ the competence knowledge. A central research issue will be to identify the possible methods of using competence knowledge so that this metaknowledge can be systematized and used by a general scheduling mechanism.

After such a general, externally directed split-responsibility scheduler is built, we will investigate the feasibility of incorporating within the system the determination of an appropriate split of responsibility between the user and the system by dynamically estimating the relative competence and response speeds of itself and the user, the difficulty of the task, and its response time requirements.

Finally, the multidimensional measure problem will be addressed by attaching "measure vectors" to knowledge and developing mechanisms for selecting knowledge to be applied based on its measure vector and for combining measure vectors of antecedent knowledge elements to produce the measure vector of the consequents.

2. PROGRAM VERIFICATION

Research Staff:
Susan Gerhart
Raymond Bates
Rod Erickson
Stanley Lee
David Thompson
David Wile

Research Assistants:
David Taylor
Jeannette Wing

Support Staff:
Lisa Moses

2.1 PROBLEM BEING SOLVED

In its most restricted sense, program verification is the task of proving the consistency of a given program with its given specifications (what is often called the "correctness" of the program). The purpose of verification may be either (positively) to establish consistency for certification purposes or (negatively) to discover inconsistency as a step in debugging.

In a broader sense, program verification is an analytic and synthetic tool for addressing a wider range of software quality considerations than just the consistency of the final specification and final program. The trend that has been developing in the last few years is to apply these techniques early in the development *process* to objects that are more like statements of requirements and designs than like programs. Indeed, particular verification methods can provide the basis for various software development methods, e.g., the data abstraction or state transition models that will be discussed shortly.

The term "verification" in this context applies to techniques involving mathematical proofs, as opposed to testing. Although testing is a much older and far more widely practiced verification method (and is usually quite effective in revealing many errors), it lacks sufficient theory to provide any form of certification [22]. Nor does testing integrate well into the early stages of software development.

Thus, the problem being solved is two-fold: (1) the specific problem of finding and refining the appropriate mathematical proof methods and (2) the more general problem of integrating these methods with the software design and development processes.

A special aspect of the research is that these mathematical and programming methods should be supported with mechanical tools which are, at the very least, capable of removing much of the burden of tedium and error-proneness from the human users. Since our goal was not to develop a theorem prover which would work completely automatically, the interface with the human directing the proof was critically important. However, we see the main goal of verification research as the development of the significant problem-related concepts, theories, and methods, without letting the underlying technology become the driving force. The development over the past decade of mechanical (even highly human-directed) theorem provers has been, and remains, a significant challenge to computing theory, artificial intelligence, and software systems.

2.2 GOALS AND APPROACH

In 1979, based on several years' work, the Program Verification Project produced a usable verification system, ***Affirm***. In 1980, we experimented extensively with it, supported outside users, and applied the system, jointly with the Internet Concepts Research group, to a major effort in communication protocols.

2.2.1 Background

In the earlier years of Program Verification (PV) research (the late 1960's and early 1970's), the primary objects of interest were programs, usually familiar mathematical ones such as Greatest Common Divisor or simple sorting algorithms. The primary areas of study in programming methodology at that time were control structures and programming languages. The only widely known method for verification was the inductive assertion method [10] with its associated intellectual challenge of devising the proper inductive assertions for loops, usually after the program's development. Following this paradigm, the system XIVUS [19] was produced by the PV project.

However, in the middle 1970's, computer science interests switched to data structures (influenced strongly by Guttag [23] and Alphard [56]) and methodologies emphasizing specifications. The PV project at that time became heavily involved with this new interest in data structures and developed a predecessor to ***Affirm*** known as DTVS (Data Type Verification System) [42]. The data structure approach seemed both promising enough and sufficiently different from other verification projects to warrant in-depth study. Thus ***Affirm*** evolved with both the "standard" program verification methods using the inductive assertion method and with newer, more data-structure-oriented methods. The present state of ***Affirm*** reflects the bias toward data structure specification and verification techniques, which have since evolved into even more general methods.

The current approach can be characterized as emphasizing the development of specifications (usually executable, but not meant to replace programs) in several levels and proving the necessary correctness properties at each level. The levels need not progress all the way down to programs, thus permitting greater flexibility of specification and verification.

The rest of the report will treat three specific areas: (1) data structuring methodology, (2) specification techniques, and (3) ***Affirm*** system development and experience. Further details on ***Affirm*** appear in [12, 43] and the PV section of [53].

2.2.2 Data Structuring Methodology

Following Guttag [24, 25, 26, 27, 28] quite literally, ***Affirm*** treats data types as abstract mathematical concepts defined by a set of axioms. A data type is defined by its primitive *constructors* which yield all data objects of the type. Other nonprimitive operations into and out of the type of interest (called *modifiers* and *selectors*, respectively) are defined by axioms for each constructor. Thus there is a systematic procedure for specifying a data type: (1) Identify the constructors; (2) Axiomatize the operations of interest in terms of the constructors.

As in mathematics, axioms are the nucleus of a theory of the concepts specified by the axioms. The most basic theorems are proved by induction on the constructors, a technique called *structural*

or *data type* induction. The typical proof of a problem-domain theorem then uses these derived properties of the subsidiary data types rather than going directly back to the axioms.

Some of the intriguing questions of the research on abstract data types were the appearance of specific types, their complexity, differences, and similarities; and the difficulty of creating them.

The actual use of the axioms in ***Affirm*** goes beyond just the specification of data type operators. The axioms look like equations but are really treated as rewrite rules in the theorem prover: whenever the pattern on the left-hand side of an axiom matches an expression, that expression is replaced by the right-hand side. Of course, this requires some restrictions on the form of the equations to prevent infinite or nonunique rewriting, but it turns out that these conditions can be checked to a large extent [44]. Rewrite rules permit a straightforward reductive brand of theorem proving.

Another interesting question in the early stages of the research was how much harder it might be to create and understand the axioms because of this additional use for theorem proving.

Of course there is more to abstract data types than just the omission of unnecessary detail in their presentation. They must be implementable in existing programming languages and in levels of abstraction. ***Affirm*** currently has no built-in methods for relating these levels, but the notion of an *abstraction* or *rep* function [29, 56] is easily expressed and used. In this approach, one states a mapping from concrete to abstract types and the appropriate correspondence between what the levels are supposed to do.

2.2.3 Specification

There is no single universal specification method. Nor is there agreement on the degree of formality desirable in specification methods or individual specifications. The basic research questions for any method are determining its range of applicability and ironing out as many difficulties as possible in its underlying theory and supporting tools.

Affirm's provisions for axioms and definitions constitute a kind of specification language. Specifications may be stated in terms of abstract functions, procedure headers with preconditions and postconditions, or abstract programs. It was a pleasant surprise to find that our algebraic axiomatic method works on far more than just abstract data types. In fact, it is a fairly general approach to specifications. In the axiomatic method, the state (of the program, or system, or whatever is being specified) is usually implicit. That is, the specifier does not say "here's what the state looks like," but instead loosely describes its components. The *state transition method* is a very popular way of specifying a large class of objects. The basic idea is to show how the state components change with respect to whatever inputs, or commands, or events are being modeled. It turns out to be easy to express this in ***Affirm***'s axiomatic framework. Suppose the state components are $\{s_i\}$ and the events or inputs are $\{E_i\}$. One simply gives the set of axioms

$$s_i(E_i(s, a), b) = c$$

where s represents the state, *b* designates the arguments to the selector, *a* shows the arguments to the events, and *c* is the change to s_i effected by E_i. Note that E_i is treated as a constructor of the data type and s_i is treated as a selector. For example, supposing the history of events at some point is s,

```
Sent(Send(s,m)) = if isEmpty(Pending(s))
                  then Sent(s) Add m
                  else Sent(s)
```

says that Sent, a queue of messages, changes in the new state Send(s,m) with the addition of a message m only when there are no Pending messages in the current state. Another way of expressing the same thing is with an explicit state constructor

$$const(v_1,...,v_n)$$

where the $\{v_i\}$ are variables of the same types as $\{s_i\}$ and with axioms

$$E_i(const(v_1,...,v_n), ..) = const(v_1',...,v_n')$$

where the v_i' are expressions giving the new state components. The latter method requires fewer, but far more complex axioms. In the former method, there are usually a large number of "no change" axioms which clutter the presentation of the data type specification.

As with data types, there must be ways to relate levels of state transition specifications. The approach is similar to that of data types using *rep* functions between states and proving that the axioms of the more abstract type are actually theorems of the more concrete level. At the level of programs, the Pascal-like language incorporated in ***Affirm*** does not support concurrency, but in many cases the lowest level processes can be represented as procedures viewed as running independently.

The state transition method expressed in ***Affirm***'s data types was the approach used in specifying communication protocols. Protocols may be modeled as distributed systems. The interesting events originate from user actions such as Sending a message, from service process steps such as Receive or Transmit along a channel or to a user, and spontaneous faults of the channels such as Losing a packet or an acknowledgment. The state of such a system consists of the data structures of the separate processes, the sequences of packets in the channels, and the histories of messages sent and received. Axioms specify, for example, how a channel between a Sender and a Receiver process changes under the pertinent events of sending and receiving packets and of spontaneous loss or reordering of packets in the channel. The goal was simply to discover what range of protocols and properties could be specified in this way and how these would be verified.

Thus ***Affirm*** affords a range of specification methods, the state transition method having been explored over the past year.

2.2.4 System

Affirm's underlying program processing component is the traditional verification condition generator for a Pascal-like language [37, 38].

The theorem prover is, in part, a descendant of an earlier prover designed by Bledsoe and Good [5, 19]. Besides using the rewrite rules of data types, the usual operations of propositional logic are built-in. The result is that the user creates proofs much like those in mathematics, using case analysis, lemmas, and various forward and backward kinds of reasoning. However, using the ***Affirm*** theorem prover is an arduous task, especially since the theorems one attempts to prove are typically not correct, but simply conjectures which require refinement to the status of theorems. The best-laid plans for proofs often go astray and the user is frequently swamped by the size of intermediate propositions or the many dangling paths taken through a proof.

For these reasons--the very nature of the mechanical theorem-proving task, as well as some of *Affirm*'s current weaknesses--heavy emphasis has been placed on the user interface. The programming of good user interfaces is not well understood, nor is there a consensus on what makes a good one (although poor interface features are readily recognized). Our experience is that a good user interface has two separate aspects: concrete and abstract. Features of a concrete user interface include command syntax, detection and provision for correction of command errors, spelling correction of command parts, profile switches, display control and quality, and interaction with the host operating system. The abstract user interface is the model of the system provided to the user, and, conversely, of the user to the system.

Affirm has a wealth of concrete features, mainly implemented from Interlisp. In each case, the introduction of the feature required considerable use and modification before it reached effectiveness; user interface features usually are not designed right the first time. This requirement for sustained usage and adaptation is probably a major reason for the poor quality of many user interfaces which are added on at the end of development without time to mature. Our approach has been to build these in gradually and experimentally.

The *Affirm* approach to abstract user interfaces is the abstract machine structure upon which the *Reference Manual* [50] is based. These include the Specification and Theorem Proving machines, which are accessed via the Executive, and which use the Formula Input/Output, Logic, and Rewriting Rule machines. Although the internal structure of *Affirm* is not actually that well-structured in terms of these machines, the documentation is carefully organized to smooth out the inconsistencies in an attempt to give the user a coherent model of the system. Thus in our conception of user interface quality, the documentation plays a major role. Ours also includes considerable analysis of experience [1, 16, 17], dedicated to transferring that experience to less experienced users. The intended effect is that users who follow the guidelines of the documentation will develop styles consistent with those we have painfully evolved over the past few years.

One of the most important facets of *Affirm*'s abstract user interface is the proof forest [9] by which proofs are developed and presented. The many functions on this proof forest give the user the feel of working on a data structure representing the way proofs are conceptualized, that is, with an overall tree structure but linked via levels of lemmas.

2.3 SCIENTIFIC PROGRESS

2.3.1 Data Structuring Methodology

The question of the appearance of abstract data types is quite well answered. The axioms follow several common patterns of recursion which also interact well with the theorem prover, although it is frequently necessary to side-step the rewrite-rule formulation to express other kinds of recursion and to package definitions which are intended to be invoked only at certain points in the proof. Indeed the many specifications of many data types are sufficiently standardized that *Affirm* provides them in a Type Library which is treated as an integral part of the system.

There seems to be no degradation in intelligibility of the axioms due to their role in theorem proving. Instead, the ad hoc quality of such axioms is removed by their entry into *Affirm* and their readability is improved by such simple mechanisms as pretty-printing and use of various fonts automatically provided by the *Affirm* user interface.

The importance of systematically developing the theory of the data type has been empirically shown again and again in experiments starting from just the axioms. If a program or another specification uses two operations of the type, there is a high likelihood that a lemma will be required that relates the two operations. The theory of the data type is this collection of properties which follow nontrivially from the axioms or from previously proved properties. Furthermore, the practical importance of systematically developing these theories before the type is used is two-fold: (1) developing the theory past the axiomatic level provides a good check that the data type operations as specified capture the intent of the specifier and perform well in theorem-proving and (2) developing the theory piecemeal is extremely error-prone because it requires the user to switch contexts between the problem of interest and that of the auxiliary data structures being used. Although not all types in the Type Library have fully developed theories, we have several good examples of what such theories look like. Similarly, the *rep* functions mapping between levels of types follow several common patterns and have their own associated theories as well [18].

We consider data abstraction to be a well-understood area, although the elaboration of specific data types and their theories remains to be done.

2.3.2 Specification

We now have considerable experience with the specification of a wide range of tasks:

- security kernels [13, 39, 40],
- protocols [2, 3, 47, 48, 51],
- distributed file updating [11, 32],
- numerical analysis bases [14, 31],
- and other programs and data types [15, 35, 55].

An interesting specification experiment is described by Lee [32]. We were given a description of an algorithm for controlling versions of files in a network [46], expressed in terms of graphs. It was first necessary to develop the graph data type and its theory in order to state the main theorem. However, attempts to prove what was claimed to be a theorem failed, leading to a counter-example which revealed a flaw in the algorithm. In this case, not even an algorithm was directly being verified, only the properties of a graph representation of a network maintained by the algorithm. This provides an excellent example of verification applicability and effectiveness very early in the design process.

With respect to communication protocol specification and verification, considerable progress was made using the state transition approach. Safety properties (that messages not be lost or misdelivered) were the primary focus of attention, but some advances were made on liveness properties (being able to respond and to do useful work). The appropriate structuring of specification levels and mappings between them was the most difficult problem. Writing axioms for the state transition machines in a single level is more straightforward. A comprehensive paper describing the techniques has been submitted for publication [51]. The section of this Annual Report describing the Internetwork Concepts Research project (7) discusses the specific protocol experiments in more depth. Schwabe [48] suggests a specification language for distributed systems, such as protocols.

2.3.3 System

Affirm was developed as a research tool to explore the theory and methodology associated with abstract data structures and, secondarily, with a Pascal-like programming language. A considerable amount of insight is gained from a tool which works well enough that its designers can perform some small-scale experiments, not necessarily to completion. However, a much more powerful, robust, and finely engineered tool is required for users other than designers, on medium- or large-scale experiments, as far through the experiment as resources permit.

In 1979 and 1980, ***Affirm*** was brought into the latter category. It was "released" over the ARPANET in December 1979, backed up by a five-volume Reference Library [49] which was completely revised recently [52]. Through 1980, several users became familiar with the system and some pursued what we would consider medium-sized experiments, primarily in the protocol area. A notable nonprotocol area is the toy security kernel [39] which led to a comparison of an ***Affirm*** proof with an HDM proof [40]. The conclusion was that the two systems were complementary, HDM (the SRI methodology) proving various data flow properties and ***Affirm*** proving several needed invariants. Another comparative study in the security area is [7]. General surveys of existing verification systems have appeared in [8, 36, 41].

Since few research tools reach even this low level of use and experimentation, it is worth trying to summarize some of the experience with it. ***Affirm*** is unique among verification systems with its orientation toward abstract data types. In comparison with other systems it is weaker than the programming-language-based methodology of GYPSY [20], than Boyer-Moore's [6] elegant and powerful theorem-proving techniques, than HDM [33] and [4] and INA JO [34] with respect to explicit methodology for supporting levels, and Oppen and Nelson's [45] specialized theorem-proving techniques. However, its design has taken into account these weaknesses (most of which are more relative than absolute) and attempted to overcome them with a user interface which makes the system, overall, easy to use.

Another strong point of the system is its emphasis on re-usability of data types and theories as supported by a Type Library. We agree strongly with the forceful arguments of Good [21] that "the problem with verification is computing science": the dominant cost in verification is in developing the deductive theories of the problem domains.

Our productivity has improved considerably over the past year, based on improvements primarily in the user interface, but the main barrier is still resources. Few verification experiments are completed when the needed cycles are only available in off-hours.

2.4 IMPACT

2.4.1 Data Structuring Methodology

Affirm has taken a piece of computer science theory and, in five years, transferred technology embedding it to a variety of application areas. The good fit between theory and technology has given impetus to the prospects for a sound and strong programming methodology based on abstract data types [30]. The tie-in with transition systems has expanded the utility of the basic axiomatic formalism, providing a uniform specification language for far more than just data types.

Other areas of impact are in the languages, especially Euclid and Ada, developed concurrently with the theory and technology.

2.4.2 Specification

The combination of abstract data types, state transition methods, and communication protocols is unique. The long-popular state transition methods have gained some mechanical support that may be applicable in a variety of application areas.

It is important to emphasize that the protocol work of the past year has been done jointly with the Internet group at ISI. This has achieved some technology transfer from verification to an application area and methodology transfer in both directions.

2.4.3 System

Affirm is one of the best known verification systems and the only one to carry through the data abstraction methodology and to emphasize user interfaces. Although verification technology has been slow in developing, it has now reached the stage where significant experiments can be performed. There are even some government contracts that stipulate verification. The specific impetus of the recent achievement of operational, even usable, verification systems is to make verification a real possibility in the next few years.

2.5 FUTURE WORK

Affirm is a stable, well-documented research tool. Its future use is anticipated in a variety of applications, including the protocol area explored this past year. Of course, numerous improvements in system, theory, and methodology are possible and will appear with its further use. Examples are a front-end to help generate the particular kind of specifications occurring in state transition models, a stronger consistency check on data types, implementation of one or more interlevel mapping methodologies, integration of a more powerful method for managing proof development histories [54], and incorporation of other theorem proving strategies such as cooperating decision procedures [45].

The particular focus of the project will enlarge the scope of programming methodology goals to adaptable software components. These are pieces of packaged software meant to become part of an assembly, perhaps with some modification. We hope to expand the role of previously developed verification and specification tools from their certification and explanation, which are clearly required for widely usable software, to include analytic and synthetic tools in the identification, packaging, and assembly of these components.

REFERENCES

1. Bates, R., and S.L. Gerhart, eds., ***Affirm*** *Annotated Transcripts*, 1981.

2. Berthomieu, B., *Proving Progress Properties of Communication Protocols in* ***Affirm***, USC/Information Sciences Institute, Program Verification Project, Affirm Memo 35, September 1980.

3. Berthomieu, B., *Selective Repeat Protocol: Axiomatization and Proofs*, USC/Information Sciences Institute, Program Verification Project, Affirm Memo 36, September 1980.

4. Birman, A., and W. H. Joyner, "A problem-reduction approach to proving simulation between programs," *IEEE Transactions on Software Engineering* 2, (2), 1976, 87-96.

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

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

7. Craigen, D., and B. Pase, *Formal Verification of Programs: Report # 3, Formal Specifications and Theorem Provers*, I. P. Sharp Associates, Technical Report TR-81-5606-3, 1980.

8. Craigen, D., and W. Pase, *A Preliminary Overview of Automatic Verification Systems*, I. P. Sharp Associates, Technical Report IPSA TR 5605-80-1, 1980.

9. Erickson, R. W, and D. R. Musser , "The ***Affirm*** theorem prover: Proof forests and management of large proofs," in W. Bibel and R. Kowalski (eds.), *Lecture Notes on Computer Science.* Volume 87: *Fifth Conference on Automated Deduction*, Springer-Verlag, 1980. (Also USC/Information Sciences Institute Affirm Memo 13, April, 1980.)

10. Floyd, R. W., "Assigning meanings to programs," in J. T. Schwartz (ed.), *Proceedings of Symposia in Applied Mathematics*, pp. 19-32, American Mathematical Society, 1967.

11. Gerhart, S. L., and D. S. Wile, "Preliminary report on the Delta experiment: Specification and verification of a multiple-user file updating module," in *Proceedings of the Conference on Specification of Reliable Software*, pp. 198-211, IEEE Computer Society, April 1979.

12. Gerhart, S. L., et al., "An overview of ***Affirm***: A specification and verification system," in *Proceedings IFIP 80*, pp. 343-348, Australia, October 1980.

13. Gerhart, S. L., *Experience with the MITRE Toy Security Kernel*, USC/Information Sciences Institute, Program Verification Project, Affirm Memo 2, January 1980.

14. Gerhart, S. L., "Fundamental concepts of program verification," in *Proceedings of American Society of Mechanical Engineers, International Computer Technology Conference*, San Francisco, Calif., August 1980. (Also USC/Information Sciences Institute Affirm Memo 15.)

15. Gerhart, S. L., *A Short Blurb on Program Specification Featuring a New Example*, USC/Information Sciences Institute, Program Verification Project, Affirm Memo 34, September 1980.

16. Lee, S., and S. L. Gerhart, eds., ***Affirm*** *User's Guide*, 1981.

17. Gerhart, S. L., ed., ***Affirm*** *Type Library*, 1981.

18. Gerhart, S. L., *Josephus Circles: An Exercise in Data Structuring*, USC/Information Sciences Institute, Program Verification Project, Affirm Memo 37, June 1981.

19. 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.

20. Good, D. I., R. M. Cohen, and J. Keeton-Williams, "Principles of proving concurrent programs in GYPSY," in *Proceedings of 6th ACM Symposium on Principles of Programming Languages*, pp. 42-52, ACM SIGPLAN, 1979.

21. Good, D., "The problem with program verification is computing science," *Software Engineering Notes* 5, (3), July 1980.

22. Goodenough, J. B., and S. L. Gerhart, "Toward a theory of test data selection," *IEEE Transactions on Software Engineering* SE-1, (2), June 1975, 195-207. (Also appeared in 1975 International Conference on Reliable Software.)

23. Guttag, J. V., *The Specification and Application to Programming of Abstract Data Types*, Ph.D. thesis, University of Toronto, Department of Computer Science, October 1975.

24. Guttag, J. V., "Abstract data types and the development of data structures," *CACM* 20, June 1977, 397-404.

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

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

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

28. Guttag, J. V., "Notes on type abstraction," *IEEE Transactions on Software Engineering* SE-6, (1), January 1980, 13-23.

29. Hoare, C. A. R., "Proof of correctness of data representations," *Acta Informatica* 1, (4), 1972, 271-281.

30. Jones, C. L., *Software Development: A Rigorous Approach*, Prentice-Hall, 1980.

31. Lee, S., *A Numerical Analysis Program Proof in **Affirm***, USC/Information Sciences Institute, Program Verification Project, Affirm Memo 31, August 1980.

32. Lee, S., R. W. Erickson, and S. L. Gerhart, *Finding a Design Error in a Distributed System: A Case Study*, USC/Information Sciences Institute, Program Verification Project, Affirm Memo 40, 1981. (To appear at IEEE Computer Society Symposium on Reliability in Distributed Software and Database Systems, Pittsburgh, July 1981.)

33. Levitt, K., B. Silverberg, and L. Robinson, *The HDM Handbook*, SRI International, Computer Science Laboratory, 1980. (Three volumes.)

34. Locasso, R., J. Scheid, D. V. Schorre, and P. Eggert, *The **Ina Jo** Specification Language Reference Manual*, System Development Corporation, Technical Report TM-(L)-6021/001/00, June 1980.

35. Loeckx, J., *Proving Properties of Algorithmic Specifications of Abstract Data Types in **Affirm***, USC/Information Sciences Institute, Program Verification Project, Affirm Memo 29, July 1980.

36. Lomet, D., et al, IBM task force on provably secure operating systems, 1980. Unpublished.

37. London, R. L., "Program verification," in P. Wegner (ed.), *Research Directions In Software Technology*, MIT Press, 1979.

38. Luckham, D. C., "Program verification and verification-oriented programming," in *Proceedings of the IFIP Congress 77*, IFIP, 1977.

39. Millen, J. K., *Operating System Security Verification*, The MITRE Corporation, Technical Report M79-223, September 1979.

40. Millen, J., and D. L. Drake, *An Experiment with **Affirm** and HDM*, The MITRE Corporation, Technical Report, December 1980.

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

42. Musser, D. R., "A data type verification system based on rewrite rules," in *Proceedings of the Sixth Texas Conference on Computing Systems*, pp. 1A22-1A31, Austin, Texas, November 1977.

43. Musser, D. R., "Abstract data type specification in the ***Affirm*** system," *IEEE Transactions on Software Engineering* SE-6, (1), January 1980, 24-32.

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

45. Nelson, G., and D. C. Oppen, "Simplification by cooperating decision procedures," *ACM Transactions on Programming Languages and Systems* 1, (2), October 1979.

46. Parker, D. S., et al., "Detection of mutual inconsistency in distributed systems," in *Proceedings of the Fifth Berkeley Workshop on Distributed Data Management and Computer Networks*, pp. 172-184, University of California, Lawrence Berkeley Laboratory, February 1981.

47. Schwabe, D., *Formal Specification and Verification of a Connection-Establishment Protocol*, USC/Information Sciences Institute, ISI/RR-81-91, April 1981.

48. Schwabe, D., *Formal Techniques for Specification and Verification of Protocols*, Ph.D. thesis, University of California, Los Angeles, Computer Science Department, March 1981. (Also UCLA Technical Report ENG 8209.)

49. Thompson, D. H., and S. L. Gerhart, eds., *The **Affirm** Reference Library*, USC/Information Sciences Institute, 1979. (Five volumes: Reference Manual, User's Guide, Type Library, Annotated Transcripts, and Collected Papers; 450 pages.)

50. Thompson, D. H., and R. W. Erickson, eds., ***Affirm*** *Reference Manual*, USC/Information Sciences Institute, 1981.

51. Thompson, D. H., C. A. Sunshine, R. W. Erickson, S. L. Gerhart, and D. Schwabe, *Specification and Verification of Communication Protocols in* ***Affirm*** *using State Transition Models*, USC/Information Sciences Institute, ISI/RR-81-88, March 1981. (Also submitted for publication.)

52. Thompson, D. H., S. L. Gerhart, R. W. Erickson, S. Lee, and R. L. Bates, eds., *The* ***Affirm*** *Reference Library*, USC/Information Sciences Institute, 1981. (Five volumes: Reference Manual, User's Guide, Type Library, Annotated Transcripts, and Collected Papers; 450 pages.)

53. ISI Research Staff, *1979 Annual Technical Report*, USC/Information Sciences Institute, ISI/SR-80-17, 1980.

54. Wile, D. S., POPART: Producer of parsers and related tools, 1981. (USC/Information Sciences Institute Technical Report, in preparation.)

55. Wing, J. M., *Experience with Two Examples: A Household Budget and Graphs*, USC/Information Sciences Institute, Program Verification Project, Affirm Memo 30, August 1980.

56. 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.

3. AUTOPSY

Research Staff:
Stephen D. Crocker
Richard Gillmann
David MacQueen
David F.Martin
Hanan Samet

Research Assistants:
Jeffrey V. Cook
Craig Taylor

Support Staff:
Joyce K. Reynolds
Debby Shishkowsky

3.1 PROBLEM BEING SOLVED

The Autopsy project has been investigating translation of programs from old languages into new languages. While it is clear from both theory and practice that translation between *arbitrary* languages is not likely to succeed, the less ambitious goal of translation between "compatible" languages may be quite reasonable.

We anticipate at least two possible benefits:

1. We have developed a specific technology for translation. While the tools we have developed are neither complete nor practical enough for production work, they may provide a blueprint for development of practical translation tools.
2. In the event that programs are planned to be written in one language and translated into a different language at a later time, our tools make it possible to work out a translation strategy and thereby uncover any difficulty. Such an exercise would provide guidance on what subset of the existing language is most compatible with the new language.

3.2 GOALS AND APPROACH

A completely automatic program translation system, though ideal, is not possible with present programming languages.

- A particular piece of code may be timing dependent, and this timing dependency may not be evident from the text of the code.
- A particular piece of code may be dependent on internal machine language interfaces that are not evident from the code.
- To facilitate maintenance by humans, a piece of code may observe coding conventions that are not themselves evident from the code.
- Some constructs in the old language may not have translations into the new language.

Despite these limitations, a translation system nonetheless must automate as much as possible of the translation; otherwise its use would not be worth the trouble.

Our approach has been to develop an interactive system that contains an array of *tools* for the user to deploy in translation. So far we have focused on the following set of tools.

Direct Translator — This tool 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 [6] 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, and we have designed an interactive monitor to govern access and use of these tools. Since the user has control over the tools and may guide the translation along any desired path--including arbitrary modification of the program by hand with the editor!--one function of the monitor is to record the history of the translation. In addition, the monitor can aid the user by providing facilities for "undoing" a part of the translation or for reapplying a sequence of tools in a uniform way.

Central to the design of the tools is the Autopsy intermediate language (IL). The direct translator translates into and out of IL representations, one of the editors is designed to edit IL representations, and the verifier verifies equivalence between IL programs. A synopsis of the Autopsy IL is given below in section 3.4.

To focus this project, we concentrated on translation between a specific pair of languages, CMS-2M [5] and Ada [4].

3.3 SCIENTIFIC PROGRESS

3.3.1 System Design

The direct translator uses syntax-directed translation techniques to perform an automatic translation in four phases. If untranslatable language constructs occur, they are noted and encapsulated for later attention from the user. The four phases of the direct translator are (1) old source to parse tree, (2) parse tree to intermediate language, (3) intermediate language to parse tree in the new language, and (4) prettyprinting the parse tree into source code in the new language. An

explicit intermediate language is used to give UNCOL-style expandability [7], i.e., if a new source language is added, we will be able to translate it to any supported new language. and vice versa.

The verification tool currently under construction compares two programs in the intermediate language representation to see if they carry out the same computation [6]. It is intended that this tool be used after changes are made by hand using an editor.

The present system has been tailored for translation of CMS-2M [5] programs into Ada [4]. Extensions to translate from or to "similar" languages would be relatively easy.

3.3.2 Status

In FY79, the intermediate language was designed and the direct translator was built, tested, and measured. Details of the direct translator are given in [2, 3].

During FY80, the interactive monitor (Quincy) was developed, editors and the direct translator were installed as tools under the monitor, and a large part of the verification system was built.

The monitor "Quincy" records all transactions and provides an "undo/redo" capabiltiy. 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.

The monitor maintains an audit trail of user commands, i.e., a file listing all the commands entered by the user. The user may undo or redo any proviously executed command by referring to its command number. If desired, the audit trail may be replayed and the commands that compose it will be redone in order.

The user is given access to two kinds of tools: all the regular programs available at the executive level, such as editors, and a set of special translation tools which are Interlisp functions.

3.4 THE AUTOPSY INTERMEDIATE LANGUAGE

An intermediate language for translation is different from an intermediate language for compilation. A compilation intermediate language throws away comments, does not provide constructs for untranslatable usages or errors, can use a stack-structured symbol table, and does not have to deal with differing scope rules. Our translation IL makes provision for these things and in addition maintains a tree-structured symbol table because the entire tree is kept during the translation process. The tree-structured symbol table is provided by distributing the symbol table over the program tree.

Our IL differs in several ways from a programming language. The syntactic "sugar" provided by programming languages has been stripped away. Scope rules, binding rules for subprogram arguments, and type conversion are explicit in the IL, whereas they would ordinarily be implicit in a programming language. Our IL provides a format for communication between passes of the system. It is intended to be flexible in order to provide for future expansion to cover more of the CMS-2M language and other languages that might be incorporated into the system.

The Autopsy intermediate language is more than the abstract syntax tree. It is similar in intent to $TCOL_{Ada}$ [1], but with an S-expression syntax that does not require numbered statement labels. The IL covers only the source level of CMS-2M; machine dependent features, e.g., microcode procedures, are not considered.

In the following description of the concrete syntax of the IL, square brackets ([]) indicate optional constructs. Three dots (...) indicate optional repeats of the last construct mentioned. The solidus (/) indicates a choice among alternates. The \ and · characters are used as break characters within names. Upper case words are literals. Lower case words are place holders for other IL constructs or terminal values. The IL definition does not restrict the nesting of IL constructs.

3.4.1 High-Level Constructs

```
(COMPILE unit ...)
(MODULE name internal-name [body])
(FUNCTION name internal-name type
     [((name internal-name type IN/OUT/INOUT VALUE/REFERENCE) ... )]
     [body])
(PROCEDURE name internal-name
     [((name internal-name type IN/OUT/INOUT VALUE/REFERENCE) ... )]
     [body])
(OPEN\SCOPE [(RESTRICTS list)] [(EXPORTS list)] body)
(CLOSED\SCOPE [(IMPORTS list)] [(EXPORTS list)] body)
```

Unlike programming languages, scopes must be declared explicitly in the IL. MODULE, FUNCTION, and PROCEDURE provide naming and argument passing without defining a scope boundary. That must be done explicitly with OPEN\SCOPE or CLOSED\SCOPE. Each time a new name is declared, a unique internal name corresponding to it is provided. The internal name is used in all subsequent references. This avoids name conflicts deriving from the differing scope rules of CMS-2M and Ada, and provides explicit typing for all names.

Arguments to subprograms must explicitly identify access restrictions and the method by which arguments are to be passed. MODULEs, FUNCTIONs, and PROCEDUREs without bodies are used for mutual recursion so that their names may be declared before their use.

3.4.2 Declarations and Types

```
(DECLARE name internal-name type [init])
(TYPE name internal-name type)

(CONSTANT type)
(INTEGER (RANGE min max))
(FLOAT (PRECISION p) [(RANGE min max)])
(FIXED\POINT (DELTA d) [(RANGE min max)])
(BOOLEAN)
(CHARACTER)
(ARRAY ((min1 max1) ... ) OF type)
(RECORD ((FIELD name type [preset])/(CASE etc.)/(COMMENT etc.) ... ))
(POINTER type)
(FUNCTION arglist type)
```

```
(PROCEDURE arglist)
(MODULE)
(LABEL)
```

All variables and all types must be declared explicitly before they are used. CONSTANT is provided for typing of compile time constants. Record fields do not need unique internal names because they are always qualified by the record name.

3.4.3 Arithmetic, Relational, and Boolean Operators

The standard arithmetic operators are provided. ADD, SUBTRACT, MULTIPLY, and DIVIDE require both their arguments to be of the same type. The arguments to POWER may be of different types. Type conversion is done explicitly using CONVERT\TYPE. Variable references use internal names. The six standard relational operators are provided. Arguments must be of the same type. AND, OR, XOR and NOT are provided. They always evaluate both of their arguments. The short circuited or "jumping" versions of AND and OR are also provided by JAND and JOR. These evaluate their second argument only when necessary.

3.4.4 Structure Access

```
(SUBSCRIPT array (sub1 sub2 ... ))
(SUBFIELD record field)
```

These forms are used to reference array and record elements.

3.4.5 Statements

```
(STATEMENTS stmt1 stmt2 ... )
(ASSIGN target source)
(LABEL name internal-name)
(GOTO target)
(IF arg1 THEN arg2 [ELSE arg3])
(CASE arg ((value1 body1) (value2 body2) ...) [(ELSE default-body)])
(FOR var FROM expr TO expr BY expr [DECREASING] [body])
(WHILE expr [body])
(UNTIL expr [body])
(CALL subr [((expr [IN/OUT/INOUT] [VALUE/REFERENCE]) ... )])
(RETURN [value])
```

The above set of statement forms is provided. STATEMENTS is used to group a sequence of statements.

3.4.6 Miscellaneous Constructs

```
(COMMENT string)
(ASSEMBLY\LANGUAGE string)
(UNTRANSLATABLE\CONSTRUCT anything)
(SYNTAX\ERROR)
```

COMMENT is used to represent source language comments, both embedded and statement level. ASSEMBLY\LANGUAGE is used for CMS-2M direct code. UNTRANSLATABLE\CONSTRUCT provides a holder for other forms that cannot be directly translated, e.g., microcode procedures. SYNTAX\ERROR is used in conjunction with the error-recovery procedures of the parser to enable the system to operate even in the presence of syntax errors, and to minimize the effect of such errors on the translated output.

The IL could be used to translate other languages such as TACPOL or JOVIAL. There might be some problems with static allocation of variables versus stack allocation, but these should not be serious. Expansion to cover more complex languages such as PL/I, Ada, or ALGOL 68 would require a number of additions to the IL in order to include such things as multiprocessing, exception handling, call by name, and compile-time macros. It would be difficult to extend the IL to include languages with radically different syntax and semantics from that of CMS-2M, such as APL, LISP, GPSS or SNOBOL.

3.5 RESULTS AND CONCLUSIONS

The most straightforward idea for translating programs is to build a complete transducer that reads programs in the old language and translates each construct or sets of constructs in the old language into one or more corresponding constructs in the new language. While this approach may work in some cases, it fails when the new language is not a complete superset of the old language. In the present case, we know that there is no real possibility of translating into Ada arbitrary programs written in the several old languages.

The Autopsy project combines automatic translation techniques with an interactive system to provide the human manager complete control over the translation process. The human may ask the system to attempt translation of all or some of an old program. The human may also control the translation more intimately by specifying that a particular transformation be applied or by supplying the replacement code himself.

There are a number of potential problem areas that complicate translation. These problem areas may be divided into five categories:

1. Constructs that are untranslatable.
2. Constructs that can only be translated by using direct code in the target language output, i.e., features of the source language that have no analog in Ada.
3. Problems that exist because of vague language definition.
4. Constructs for which translation is possible, but ugly.
5. Constructs that raise philosophical issues regarding their proper translation.

We consider each of these in turn.

1. Untranslatable constructs

   Timing loops — Any source code that depends on the execution speed of code translated by a particular compiler probably will not work in the translated code.

Direct code that makes assumptions about the compiler

If the source program contains direct code (assembly language), and that code makes assumptions about the compiler (e.g., register use, calling sequences, naming conventions for variables), then the direct code itself will have to be changed.

Overrunning array bounds

Many languages allow the programmer to deliberately index an array with an index outside the declared range. Ada normally checks for this condition. Thus, any code which depends on overrunning an array would not be translated in such a way as to run correctly in Ada, without using a **pragma** to suppress the index check.

System index and local index

The system index feature of CMS-2M cannot be translated because it causes the compiler to allocate registers differently. It cannot be assumed that Ada compilers will be so compliant.

Code length

The code produced by the translated Ada program may be longer than the code produced by the original source program, which could cause an insufficient memory problem.

2. Constructs requiring direct code translation

Microprogram declarations

CMS-2M supports microcode procedures. This feature has no analog in Ada. Of course, modifying the microcode may also have unfortunate side effects that would invalidate the translation.

3. Vague definition problems

Array storage order

Ada does not dictate the storage order for multidimensioned arrays. Thus we do not know whether or not it is necessary to reverse the order of subscripts in the translation. This makes a difference in paged systems and to direct code assuming a particular storage order.

Warning messages

The compiler may accept certain erroneous inputs and yet produce correct output, usually after giving a warning message. In effect, this expands the language, but in a poorly defined way.

Undocumented features

The source language compiler may have extra features which are not documented, but which are used in programs.

4. Ugly translations

Fixed-point arithmetic

The CMS-2M manual describes an algorithm for binary-point alignment during fixed-point arithmetic. Ada uses a different algorithm that requires the specification of more detail.

Vertical format tables

CMS-2M vertical format tables are arrays of structures stored with the

indexed subfields grouped instead of the usual way. These can be translated into Ada but require a different data structure.

Different reserved words

There are reserved words in Ada that are not reserved words in CMS-2M. If these are used as identifiers in source programs, they will have to be renamed in the translation.

5. Philosophical issues

Compile-time macros

CMS-2M provides compile-time macro facilities. The issue is expanding these before the translation versus producing equivalent Ada compile-time macros as part of the translation process. We are taking the position that the string substitution MEANS macros should be expanded before the translation because they may include references to CMS-2M keywords that would have no direct counterpart in Ada. On the other hand, EQUALS macros provide compile-time variables and arithmetic, which has a direct analog in Ada, and this is translated.

Compiler bugs — Where the source compiler has a bug should it be duplicated in the translation, "fixed," or forbidden?

3.6 IMPACT AND FUTURE WORK

The basic concept of translation under user control has been demonstrated. Further work is necessary to produce a useful system, but the basic framework appears sound. A somewhat surprising result is that Ada code produced by the translator is no larger than the source CMS-2M code.

This work has ceased at ISI, but a group at the Aerospace Corporation interested in J73 to Ada translation is continuing work with this system.

REFERENCES

1. Brosgol, Benjamin M., et al., *$TCOL_{Ada}$: Revised Report on An Intermediate Representation for the Preliminary Ada Language*, Carnegie-Mellon University, Technical Report CMU-CS-80-105, February 1980.

2. Crocker, Stephen D., and Richard Gillmann, "AUTOPSY: Conversion of CMS-2M programs to Ada," in *Proceedings of the CMS-2 User Group Conference*, Fleet Combat Direction Systems Support Activity, San Diego, Calif., February 1979.

3. Gillmann, Richard, Stephen D. Crocker, and Craig Taylor, *Translation of CMS-2 Programs to Ada*, USC Information Sciences Institute, WP-19, February 1980.

4. Ichbiah, Jean D., et al., *Reference Manual for the Ada Programming Language*, United States Department of Defense, 1980.

5. *User's Handbook for AN/UYK-20(V) Computer, Support Software, CMS-2M*, NAVELEX, 1975.

6. Samet, Hanan, "Proving the correctness of heuristically optimized code," *Communications of the ACM* 21, (7), July 1978, 570-582.

7. Steel, T. B., "A first version of UNCOL," in *Proceedings of the Western Joint Computer Conference*, pp. 371-377, AFIPS, 1961.

4. FORMAL SEMANTICS

Research Staff:

Stephen D. Crocker
Avra Cohn
Richard Gillmann
Michael Gordon
David B. MacQueen
David F. Martin

Research Assistants:

Robert Mcdonnell

Support Staff:

Joyce Reynolds
Debbie Shishkowsky

4.1 INTRODUCTION AND BACKGROUND

The principal goals of this project are the development of tools and methodologies for supporting the development of precise, readable, and accurate formal semantic definitions (FSDs) of programming languages. The specific research focus of the project is building tools for manipulating, processing, implementing, and testing the formal definition of Ada written by a group at the Institut National de Recherche en Informatique et en Automatique (INRIA) in France.

The ISI Formal Semantics Project grew out of a pre-existing effort (the Autopsy Project) to develop translators between source languages. Part of that effort was theoretical and resulted in our learning a great deal about denotational semantics. In addition, the Autopsy Project focused on translation of CMS-2M programs to Ada, which brought us into the Ada orbit. A month-long visit by Gilles Kahn and Veronique Donzeau-Gouge of INRIA in August 1979 opened up communication between the ISI and INRIA groups.

At the Ada Test and Evaluation Workshop held in Boston in fall 1979, the Ada FSD was discussed and preliminary drafts were made available. Because of the complexity of this definition, machine aids for manipulating and verifying it were believed to be highly desirable, and this project was initiated.

4.2 FORMAL DEFINITION OF ADA

DoD commissioned the design and implementation of the Ada programming language with the intention of requiring most future military systems to be programmed in Ada. It is therefore necessary that Ada be precisely understood by both its users and implementors, in order to ensure the quality of systems written in Ada. In particular, since DoD must control Ada compiler implementations, a precise, well-structured, and validated formal definition of Ada can provide one of the principal standards to which these implementations must adhere.

4.2.1 The INRIA Definition of Ada

A denotational formal semantic definition (FSD) of Ada [2] has been developed at INRIA [1]. Of the various techniques for writing formal semantic definitions, denotational semantics was chosen by INRIA for its conceptual clarity, strong mathematical foundation, conciseness, and implementation-

independence. There is some experience in writing definitions of real languages using denotational semantics (e.g., Algol 60, LISP, SNOBOL, Pascal, Algol 68), and there are some tools available which support this form of definition (e.g., SIS [3]). Despite its conceptual power and popularity, however, denotational semantics (in common with other semantic notations) exhibits weaknesses in practice when applied to "large" languages like Ada. A typical denotational semantic definition consists of a collection of mutually recursive functions, which, in the case of Ada, is quite large. Understanding the semantics of an Ada construct requires the application of a generally long sequence of these functions to an abstract syntactic representation of the construct. Attempting to do this without machine assistance will most certainly result in a great many errors and is, in a practical sense, impossible. Consequently, a complete understanding of the Ada FSD is difficult to achieve due to its sheer size and complexity; unaided human application of this FSD to understanding Ada programs is at best an arduous task.

4.2.2 Processing the Ada FSD

The preliminary *Ada Reference Manual* [2] is divided into a number of chapters, each devoted to a specific aspect or component of the language. None of the FSD appears in this manual. The draft Ada FSD [1], however, is organized by "folding" it into the manual so that each chapter contains the pertinent procedures of the Ada FSD.

The draft Ada FSD is available on the ARPANET (at HI-MULTICS); all our work is done on copies of this source. The text of the FSD contains embedded format and font control commands, intended for processing by a text-formatting system. We have developed tools for processing this raw text, preparatory to the main task of implementing and testing the Ada FSD, among which are:

- a program to automatically translate the embedded format and font control commands into equivalent commands for the Scribe document production system [4], so that formatted copies of the FSD can be produced at ISI;
- a program to extract the procedures of the FSD from the text they are embedded in;
- a program to translate the procedures of the FSD into an intermediate form that serves as input to other tools;
- a cross-reference program for the Ada FSD.

4.2.3 Development of an Executable Ada Formal Definition

It is imperative to construct appropriate tools to aid the understanding and validation of the Ada FSD. Such tools could be used in two ways. Initially, Ada test cases whose semantics are well understood could be used to test the correctness of the FSD. Subsequently, after confidence in the correctness of the Ada FSD has increased, the tools could be used to answer specific questions about parts of the FSD as they relate to example Ada programs whose semantics are not readily apparent. A plan has been developed for validating and understanding the Ada FSD, using supporting tools whose design and implementation are proposed.

Other tools for machine-processing and testing denotational semantic definitions exist, principal among them being Peter Mosses' Semantics Implementation System (SIS [3]). Serious consideration was given to using SIS, but translating the Ada FSD into a form processable by SIS did not seem to be an attractive course of action because:

- the procedures of the FSD would have to be automatically translated into DSL (SIS's semantic metalanguage), and it was not clear that this task would be straightforward;
- operation of the lambda calculus interpreter underlying SIS cannot be controlled, and run-time errors committed by an interpreted formal definition are very difficult to diagnose;
- the inner workings of SIS are not documented to an extent that makes SIS readily modifiable, should that become desirable or necessary.

The Ada FSD is written in an "Ada-like" sugaring of typed lambda calculus; we call this language AFDL, an acronym for Ada Formal Definition Language. Our validation plan consists of translating the procedures and data types of the Ada FSD into an interpretable representation intermediate language (AFDL-IL), transforming candidate Ada test programs into abstract syntax trees, and then applying the translated FSD to the abstract syntax trees to obtain the (static and dynamic) semantics of the corresponding programs. The semantics thus obtained can then be compared to the expected semantics. In addition, tools to generate useful items such as cross references of the FSD's procedures and intrinsic data types will be designed and implemented.

Despite the apparent straightforward nature of this plan, there are many problems that must be solved in order to implement it. The plan given below outlines these problems in an orderly framework. First, to set the stage, an overview of the structure of the Ada FSD is given, followed by a review of the status of the Ada FSD. Then the proposed plan is outlined, and some of its principal tasks are discussed.

Structure of the Ada FSD

Basically, the Ada FSD consists of a collection of mutually recursive value-returning procedures together with a repertoire of basic data types. From an operational point of view, the Ada FSD is organized into three "phases," one of which is syntactic and the others semantic. The syntactic phase establishes a relationship between the concrete and abstract syntax of Ada by providing a specification of both the concrete and abstract syntactic domains together with a (constructive) mapping from the former to the latter. In practice, this mapping is implemented as a parse-driven construction of Ada abstract syntax trees from corresponding Ada program strings. The semantic phases, which process abstract syntax trees, are two in number. The first, called static semantics, performs what are generally considered to be "compile-time" functions such as static type-checking. If this phase fails to find any errors, then it produces an abstract syntax tree which is a modified form of the tree input to the static semantic phase; otherwise, an error message is output. The final semantic phase, called dynamic semantics, determines the "run-time" semantics (the meaning of procedures, expressions, etc.) of statically checked abstract syntax trees output from the static semantics phase.

Status of the Ada FSD--September 1980

At this time, the Ada FSD is incomplete in two major respects: (1) AFDL is not completely characterized, and (2) some of the FSD is missing. Consequently, one of the first orders of business is to remedy these discrepancies. AFDL must be appropriately characterized; this can be done as a joint effort by ISI and INRIA. Procedures missing from the FSD must be identified and supplied; the burden of supplying the missing procedures necessarily falls on INRIA. Our perception of the status of the Ada FSD as of September 1980 now follows. Emphasis is placed on the status of the definition of Ada abstract syntax, the definition of the basic data types of AFDL, and the degree of completion of the AFDL procedures constituting the static and dynamic semantics of Ada.

ABSTRACT SYNTAX

In the *Reference Manual* [2], the concrete and abstract syntaxes of Ada are defined, the latter less explicitly than the former. In fact, there exist two abstract syntaxes: a post-parse abstract syntax (which is input to the static semantics phase) and a post-static-semantics abstract syntax (which is input to the dynamic semantics phase). The correspondence between the (post-parse) abstract syntax and the concrete syntax given in the Reference Manual is implicit; it must be given explicitly or else the reader must be given enough information to deduce the exact correspondence, as this is necessary for a detailed understanding of the FSD. The form of the abstract syntax is currently being revised by INRIA.

BASIC DATA DOMAINS OF THE FSD

The FSD is written in a language that is said to be an "extended subset of Ada." The procedures of the FSD use several basic data types (trees, environments, continuations, etc.), which are not declared or otherwise defined. If indeed the FSD is considered to be written in an Ada subset, then these basic data types must be declared or "packaged" in order to make the FSD a complete subset-Ada program. If on the other hand the FSD is considered to be written in a syntactically sugared version of typed lambda calculus (this is the view we favor), then these basic data types can be considered to pre-exist, and thus they would be a "hard-wired" part of the definition (and thus of the semantics of AFDL). They must still be defined, however.

AFDL SEMANTICS

If AFDL is regarded as sugared lambda calculus and if its basic data types are predefined as mentioned above, then deducing the semantics of AFDL presents no significant difficulties. The only novel nonlambda-calculus features of AFDL are a case statement and the specialized use of Ada generics, which can be compiled into "standard" lambda calculus equivalents, along with the overloading of some operators.

FSD PROCEDURES

Most of the procedures of the Ada FSD seem to be present and complete. However, a more accurate estimate can be made only with the aid of a mechanized cross-reference, which can be generated by appropriate tools.

The Plan

The Ada FSD must be completed in accordance with some of the above discussion, and AFDL must be characterized and implemented. In order to test the FSD, its syntactic and semantic phases will have to be implemented. Prior to that, the mapping between concrete and abstract syntax trees will have to be more explicitly specified than it is in the draft Ada FSD. In fact, the concrete syntax should be defined using a grammar more suitable for parsing (such as an LR(1) grammar) than the concrete syntax given in the manual. Some of this has already been done, and the rest can be completed by ISI with some interaction with INRIA. Implementing the semantic phases will require understanding and reconciling the semantics of AFDL and whatever executable representation is chosen, so that the required translation from AFDL to the executable representation can be accomplished. Suitable test cases must then be designed and run, and the results interpreted.

Evaluation Strategy for AFDL

The basic semantics of the Ada FSD is that of the (typed) lambda calculus, which assumes a specific order of expression evaluation (call-by-name), and static binding of free variables in expressions. If the executable translation of the FSD is to be faithful to this semantics, it follows that the execution mechanism should also use the same or equivalent binding and order of evaluation.

Since Interlisp uses dynamic binding and a different order of expression evaluation (call-by-value), it would not be easy to use the Interlisp evaluation mechanisms to faithfully model the semantics of AFDL. Call-by-name cannot be approximated very well using NLAMBDAs and similar devices (partly because of dynamic binding). A special-purpose interpreter based on an algorithm similar to that used by Mosses in implementing DSL seems to be more promising.

Type-Checking AFDL

Static type-checking of the AFDL code is an essential part of testing the Ada FSD. It also eliminates the need for most run-time type-checking during execution of the FSD. The simplest way to type-check AFDL code is to first translate the code into an intermediate form (abstract syntax) which retains the declarative type information, and then apply a type-checking function which returns a similar form with the type information removed if type-checking succeeds, and otherwise identifies the type error. The result of (successful) type checking is a form which can be interpreted directly, or translated into an equivalent Interlisp form.

Since AFDL is essentially an Ada-like sugaring of a typed, applied lambda calculus (the usual vehicle for expressing denotational semantics), we can start with a standard, fairly simple algorithm for the typed lambda calculus and add some ad hoc components to deal with exceptional cases like generic procedures and operators.

The design and implementation of a type-checking system for AFDL will involve the following subtasks.

- The basic types used in the definition must be identified, along with associated primitive operators and their types. These constitute the base case of the inductive type-checking process.
- Generic procedures and their instances must be identified and special type-checking routines written to handle them. Fortunately, it appears that only a few, fairly simple, generic procedures are used in the FSD. Typically, they are used for iterating over sequences and similar limited and well-defined purposes.

A number of other issues or potential problems must also be considered, including:

- **Representation independence of basic types.** For the most part, the basic types used in the FSD appear to be used via their associated primitive operations, so their "internal" structure or representation is irrelevant. A minor exception is the use of continuation types which have a functional structure, so that values of a continuation type (such as *EXEC-CONT*) can be *applied* to other values. Any serious violations of representation independence would complicate type-checking.
- **Type-free treatment of abstract syntax.** There is a certain looseness or impreciseness in the type structure of the abstract syntax. All syntactic structures are of the same type, namely *TREE*, and the syntactic types are indicated by a component of type *CONSTRUCT*. This means that operations, such as selectors, associated with the abstract syntax will have to do run-time type-checks on their arguments. This could be avoided by fully elaborating the type structure of the abstract syntax (i.e., introducing a separate type for each construct).

- **Overloading.** There does not appear to be any use of overloading in the FSD. If overloading were present, it would make type-checking significantly more complicated. An apparent exception is the fact that abstract syntax selectors are overloaded, but this is not true overloading because of the lack of type structure in the abstract syntax. In effect, these overloadings must be resolved at run-time along with the syntactic type-checking.

In summary, static type-checking of the Ada FSD should be relatively straightforward, but it cannot be complete because of the lack of type structure in the abstract syntax.

4.3 SUMMARY

Our research toward the design and implementation of tools and a methodology for testing the Ada formal semantic definition is well under way. We expect that our work will provide a practical means of ensuring the consistency and accuracy of the FSD, thereby making it more reliably useful to the Ada community.

During the coming year we expect to

- precisely ascertain to what degree the Ada FSD is complete;
- complete the characterization of AFDL;
- complete the "static" validation of the FSD by parsing and type-checking its procedures;
- make significant progress toward the design and implementation of the AFDL interpreter;
- implement an Ada parser that outputs the abstract syntax representations of Ada programs.

REFERENCES

1. Donzeau-Gouge, Veronique, Gilles Kahn, and Bernard Lang, *Formal Definition of Ada*, Honeywell, Inc. and CII-Honeywell Bull, 1980.

2. Ichbiah, Jean D., *et al.*, *Reference Manual for the Ada Programming Language*, U.S. Department of Defense, 1980.

3. Mosses, Peter D., *SIS: A Compiler-Generator System Using Denotational Semantics (Reference Manual)*, University of Aarhus, Department of Computer Science, Institute of Mathematics, DK-8000 Aarhus C, Denmark, 1978.

4. Reid, Brian K., and Janet H. Walker, *Scribe Introductory User's Manual*, 1980.

5. COOPERATIVE INTERACTIVE SYSTEMS

Research Staff:
William Mark
David Wilczynski
Robert Lingard

Research Assistant:
Tom Lipkis

Support Staff:
Andrea Putnam

5.1 INTRODUCTION

The recent concentration of effort in interactive systems has produced a number of powerful online services to help users solve problems. The functions performed by these services are often closely aligned with user needs. However, user acceptance of these services has been quite limited due to the lack of an adequate user interface. While trained computer users are usually willing to contend with the idiosyncrasies of individual services, office managers, military commanders, and home users are far less tolerant. Though the transfer of computer technology to these "real user" environments is now economically feasible because of recent hardware advances, the transfer will fail unless the user interface for the services on the system makes them easy and natural to use.

A "cooperative" interactive system is one which provides a natural interface for its users, responds rapidly to user requests, and behaves in a consistent manner across a wide range of functional services and input/output capabilities. Our research in this area is centered on the construction of a system, **Consul**, which will provide natural input/output and help facilities for users of interactive services such as electronic mail, automated appointment calendar, and document preparation. Although the system is intended to be used by, and be individually adaptable to, a wide class of users ranging from novice to experienced, our concentration is on the problems of the novice and occasional user.

The Consul system provides a methodology for building services into a single natural interface facility. This approach is based on our belief that no fixed set of services can satisfy the interactive computing needs of a diverse community of users. Users' needs change constantly, and vary greatly from one environment to another. We believe that the needs of any particular group of users must be satisfied by service builders familiar with those needs.

However, the task of building an interactive service that provides a natural interface and remains consistent with other services is too great to be left as a burden for individual service builders. What is needed is a system that has already solved the basic problems of providing a natural, consistent interface, and into which new services (or changes to existing services) can be incorporated with relative ease. This is the purpose of the Consul system.

A prototype system has already been built to demonstrate some of these ideas. The design and capabilities of this system will be described in detail below. An outline of ongoing and planned research activities follows, describing the extension of this experiment to demonstrate a working cooperative system in an advanced computing environment with electronic mail and appointment calendar services. Finally, the impact of such a system on future interactive computing is discussed.

5.2 THE PROBLEM

The technology for building user interfaces has not kept pace with the growing demand for interactive computer services. Interfaces to interactive services still do not follow general conventions, offer adequate help and documentation, or provide sufficient flexibility. Most interfaces do not even attempt to adapt to users, accept the user's natural input form, explain errors, or answer user questions. This is really not very surprising as user interfaces are hard to write--especially since no consistent methodology for developing interfaces has ever been formulated. The additional burden of building an interface with enough knowledge of users and systems to be truly cooperative is simply too great to allow implementation in any but a very few services.

The burden of building user interfaces is especially serious as we enter this era of new computing environment development. As new systems are built, many totally new services will be implemented in the new environments. Given the notorious longevity of services--good or bad--it would be a serious setback if the new services were constructed in the same inadequate way as the old.

The problem, then, is that interactive services are hard to use and that easier-to-use services are hard to build.

Interactive services are hard to use because:

Individual services come complete with their own conventions, command language, and capabilities. Most interactive systems, like DEC TOPS-20 or IBM TSO, offer an environment in which each user program is self-contained and responsible for all of its own user interactions. As a result, service builders tend to be independent in their approach to user interaction; consistency with other services is rarely a design goal.

Interactive services offer inadequate help, documentation, and explanation facilities. There are few examples of systems which offer useful help features as an integrated part of normal operation. Those that do often cannot offer the right *kind* of help to users. For example, the SIGMA message system [16] had a sophisticated online help system [14] that organized information about its terms as a tree-structure. However, user interviews showed that this facility was generally ignored. Users who did try the help facility did not find it helpful. The static kind of information in this kind of help data base is best presented in manuals. What users really need are answers to questions like "How do I do X?" "What happens if I ...?" "Why did Y occur?" "Can I undo the effects of Z?" To answer these questions, a system needs to have knowledge about its own functionality and its dynamic execution state.

There is an inevitable mismatch between user needs and service capabilities. Command languages, function keys, pointing devices, etc., are inevitably deficient in their ability to express all of the possible variations and combinations of users' needs. User requests may therefore go unanswered--even though the desired functionality is present in the actual service--because of a mismatch between the user's expression and the system's expectation. Currently, users resolve this mismatch by seeking aid from human experts; but experts are often unavailable.

Cooperative interactive services are hard to build because:

Cooperative interfaces are large, complex pieces of code that are hard to adapt to different services. It has been estimated that more than half the code in current interactive systems is devoted to the user interface; and according to Schwartz [15], highly interactive programs are four times as hard to write as noninteractive programs. The few attempts to build cooperative interfaces have each concentrated on enhancing *specific* services [4, 5, 6, 7]. Though leading to individually successful systems, this approach has two limitations: 1) the techniques developed tend to rely fundamentally on

the peculiar characteristics of that service--it is hard to see how to apply them to new services; and 2) the techniques often depend explicitly on having a small set of functions --they have not been applied to services with what we consider to be a "critical mass" of functionality.

Interactive services must be designed for adaptability and change. Services that require adaptation by the user will have very limited appeal. Services must therefore adapt to the needs and style of the user, especially as the user develops expertise and higher expectations. These expectations can lead to changes in the user's requirements for service functionality. Any resulting modifications must be made in the context of the user's model of the service. Changes should perturb the user's model only slightly; maintaining the style and structure of the user interface is crucial.

Service builders do not have an appropriate environment in which to build cooperative interactive services. Services in TOPS-20- or TSO-type systems cannot share functionality and cannot communicate with one another in any meaningful way. The UNIX operating system [8] attempts to address this problem by giving programmers a system language for configuring complex functions out of smaller ones. However, the arbitrarily low-level functionality with which UNIX must deal prevents it from applying any high-level conventions to the composition and communication tasks. Its function-sharing capabilities therefore provide little help in building a cooperative interface.

5.3 TECHNICAL ISSUES

The object of the Consul system is to provide an environment in which interactive services are easy to use and easy to build. What is needed is a methodology for interactive systems that

- provides a solution to the problem of building in enough knowledge about users and systems to provide intelligent flexibility in the interface,
- provides guidelines for the construction of interactive service software,
- enforces consistency at the interface to all services.

Achieving these goals requires the solution of a number of research problems:

1. Representing the knowledge that allows a natural interface: Providing natural language input and explanation facilities requires large amounts of domain-dependent knowledge represented in a consistent framework. Much of the Consul research effort involves modelling the possible characteristics and behavior of users and services. The result is a carefully structured, built-in knowledge base containing a **systems model**, a conceptual framework for and representation of possible service operations, data structures, and events.
2. Mapping between different world views to provide flexibility: The flexibility to handle diverse and ultimately unpredictable user requests requires mapping the user's descriptions of his needs into the service's descriptions of its capabilities and requirements. Consul's systems model includes the **inference rules** required to map between different forms of description. Consul's **mapper** uses these rules to transform descriptions of user requests into calls to the service functions that can be executed to satisfy these requests. If the request calls for explanation, the mapper uses inference rules to transform descriptions of system activities and errors back into the user's world to produce a response.
3. Building services to support flexibility: A natural interface requires flexibility in the use of service functionality. Services must therefore be constructed so that the granularity of their functional modules is small enough to allow a large variety of dynamic recombinations (to satisfy a large class of unpredicatable task specifications). Of course, implementation of a service as a large number of functional modules requires careful attention to problems of system organization, control structure implementation, and performance. Consul's **process script formalism** provides a mechanism for

constructing, organizing, and controlling services as structures of small-grained functional modules.

4. Ensuring consistency from the user's point of view: The user interface must not only be "natural," but *consistently* natural across services built by different individuals, providing different functionality, and satisfying different user needs. Consul provides interservice interface consistency by always structuring service-dependent knowledge in terms of its single **systems model**. This model transcends individual service boundaries and organizes all services into a single Consul knowledge base.

5.4 APPROACH

These individual responses to technical problems are actually parts of a single Consul approach to the cooperative systems problem. This section describes the approach in the context of the current system and its two major thrusts: making a service easier to use by handling the user's natural input, and making the service easier to build by providing the process script software design methodology.

5.4.1 The Current System

The Consul system consists mainly of description environments ("models") for representing the various kinds of knowledge it requires--knowledge about users, services, and interactive systems in general. Relationships between descriptions are represented in terms of inference rules, which are "applied" to transform one kind of description into another. Thus, understanding a natural language user request means redescribing it (using inference rules) until it can be seen as a service model description of a function call. This function call is then executed to satisfy the user's request.

Consul integrates new services into its environment by influencing the service-building process via its process script software design methodology. Implementing services in the Consul environment includes describing them in terms of Consul's abstract service-independent model of interactive systems. This ensures a unified and consistent treatment of service-dependent knowledge within the system. The process script programming formalism has been especially designed for the task of providing an adequate service description as part of the programming process.

The basic structure of the Consul system is shown in Figure 5-1. The knowledge base is central to all system activities--parsing, mapping, and execution. It contains several kinds of information:

Systems model: a service-independent representation of the detailed behavior of the basic operations found in any service (e.g., deletion, scheduling, display), along with the data structures these operations work on (files, tables, display lists, etc.). The particular operations and data structures of any service can be described in terms of this representation, and thus be "understood" by Consul (i.e., seen in relation to the other things that Consul knows about).

Service model: a particularization of the systems model to the actual operations and data structures of some interactive service that is implemented in Consul. For example, the model for a message service would describe *specific* (actually executable) functions like "DeleteMesssage" and "ShowMessage," and specific objects like "MessageFile" and "MessageDisplayHeader"). This service model would thus describe the actions and objects of the message system in terms of the general concept structure laid out by the systems model.

Process scripts: a programming formalism for implementing services within the Consul system. Process script programs consist of two parts: a procedure to perform some action on the machine and some descriptive information about that procedure. The

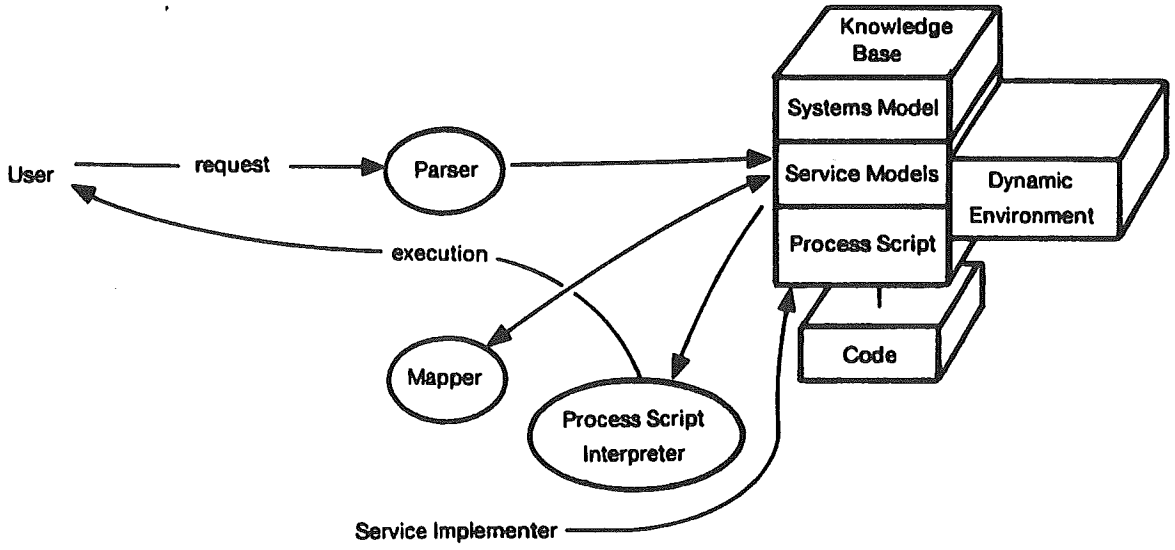


Figure 5-1: The current Consul system

descriptive part is in the form of a small number of categories of information required by Consul in order to see how the function represented by the process script fits into its systems model (see Figure 5-6).

Code: the application language programs that implement the lowest level functions of the service (e.g., how a file pointer is generated from a file name). Since it is not practical for Consul's model of an operation to extend all the way down to the most detailed level of implementation, Consul gives the programmer a mechanism for describing some aspects of his service as black boxes as far as Consul is concerned. The connection between these black boxes and the process scripts of the service is in terms of **process atoms**--identical to process scripts except that they contain, instead of a body, a call to a piece of actual application code (the black box)--see Figure 5-7. A service thus consists of a set of process scripts that relate service functions to the rest of Consul's knowledge base (through the process script descriptors) on the one hand, and to pieces of application code (through the process atoms) on the other.

Dynamic Environment: a model of all system and user activities as events in time, i.e., invocations of the actions defined in the user and service models. This event model serves as a dynamic environment for expressing the *behavior* of the user and the system.

All of these models are currently under development in the Consul system. The systems model, which has been the focus of much of our attention, contains a spanning set of abstract actions and objects, with detailed structure for a number of them. A service model is currently being constructed for a message system [16]. An initial definition of the process script language has been formulated, and about 25 message system functions have been written in terms of it.

Using this knowledge base to provide interface facilities (handling natural language, explanation, error handling) is a process of knowledge-based inference and process script execution. The system's basic operating mode is as follows: the user types a request into the system; the request is parsed, i.e., rendered into the system's knowledge representation; inference is used to map the representation of the user request into a description of some system action; this action, a service operation, is then executed by the process script interpreter, thus fulfilling the user's request. These mapping and execution processes are outlined below and then discussed in the context of a real scenario of Consul action.

Parsing: A very powerful parser [1, 2] has been interfaced with the current Consul system. The parser is capable of handling a wide variety of requests, and the Consul knowledge base discussed above will provide the underlying semantics to allow it to do so. For example, the parser's interpretation of the request

Get rid of message 6.

would be constructed as a specialization of Consul's knowledge base representation of "requests to remove things." That is, the parser translates the user's natural request form into the system's knowledge base form. Parser development *per se* is not part of our research effort, but is being carried out cooperatively at Bolt Beranek and Newman, Inc.

Inference: Parsing the user request is not the same as understanding how to respond to it; "understanding" is achieved via rule-based inference [10]. Each input to the system is first classified in the knowledge base according to its relationship with the knowledge that is already there. However, the initial classification produced by the parser is rarely sufficient to allow the system to take action (i.e., there is no "get rid of message 6" operation). The user's request must therefore be redescribed (and reclassified) if the system is to know how to handle it. Redescription is accomplished by the application of inference rules. For example, a rule exists to redescribe "requests to remove things" as "calls to delete operations." If this rule is applied to the user's initial request, the request becomes a "call to a delete operation on message 6." This redescription is then classified in the knowledge base. Rule application continues until a description of a call to an executable operation (i.e., an actual process script--in this case, the one to delete messages) is generated. This call can then be executed to satisfy the user's request. The scenario in the following section shows the mapping process in detail.

Execution: Process scripts are executed interpretively in a special environment set up by Consul. Process script execution involves more than evaluation of the procedure body to achieve a functional effect: the process script interpreter must access the dynamic environment to make processing decisions and update this environment to show execution results. Also, when a process atom is executed, the interpreter must invoke the appropriate application code in the processing environment of the application language.

5.4.2 Making Services Easier To Use: Handling Natural Requests

This section gives a step-by-step account of Consul's processing of the user request

Show me a list of messages.

for a message service based on the SIGMA system [16]. The scenario describes the system's mapping of this request into two process script calls,

OpenSummaryFile;

ShowSummaryFile;

which are then executed to satisfy the request.

The first stage of this mapping effort is translating the input utterance into Consul's network representation, as implemented in the KL-ONE formalism [3]. This is the job of the parser, which, based on the system's model of "show requests," produces the KL-ONE concept **Show Request.1**, shown in Figure 5-2 (assuming that the request came from user "Smith").

Consul classifies this structure in its knowledge base, finding, as shown in Figure 5-3, that it is a subconcept of the condition of **Rule1**. This means that Consul can redescribe **Show Request.1** as a call to a "display operation" according to the conclusion of **Rule1**. The result is a new description,

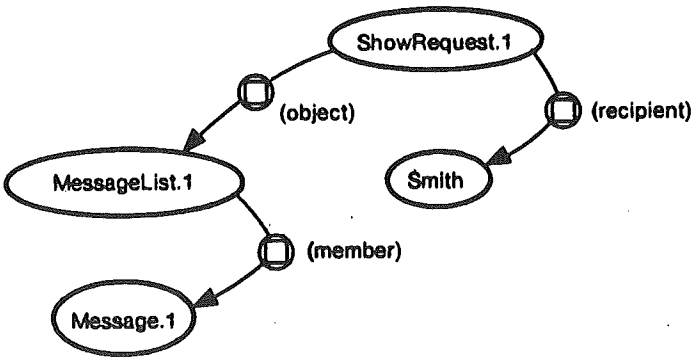


Figure 5-2: The parse of "Show me a list of messages."

Display Operation Invocation1.1, which Consul then classifies in its knowledge base. If, via this classification process, the new description is found to be a subconcept of an executable function, inference is complete--**Display Operation Invocation1.1** can simply be passed on to the Consul interpreter. Otherwise Consul will have to use additional rules to refine the description until it can be seen as an actual call on some other function or functions.

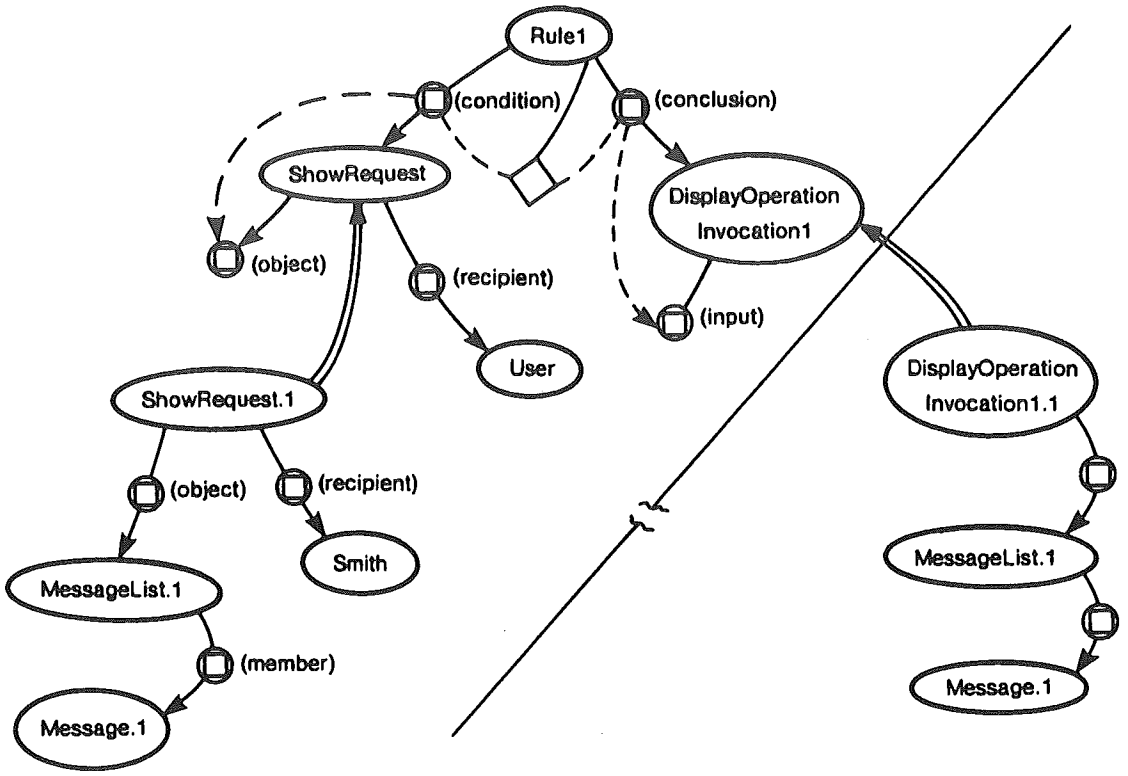


Figure 5-3: Application of Rule 1

Figure 5-4 shows the classification of **Display Operation Invocation1.1** in Consul's knowledge base. The description is not a subconcept of an executable function (flagged executable). This means that Consul must find an applicable mapping rule to further redescribe **Display Operation Invocation1.1**. Figure 5-4 also shows that **Display Operation Invocation1.1** has been classified

as a specialization of **Display Operation Invocation2**. A fuller view of the knowledge base, seen in Figure 5-5, shows that **Display Operation Invocation2** happens to a mapping rule condition. This means that the rule, **Rule2**, is applicable to the current description **Display Operation Invocation1.1**.

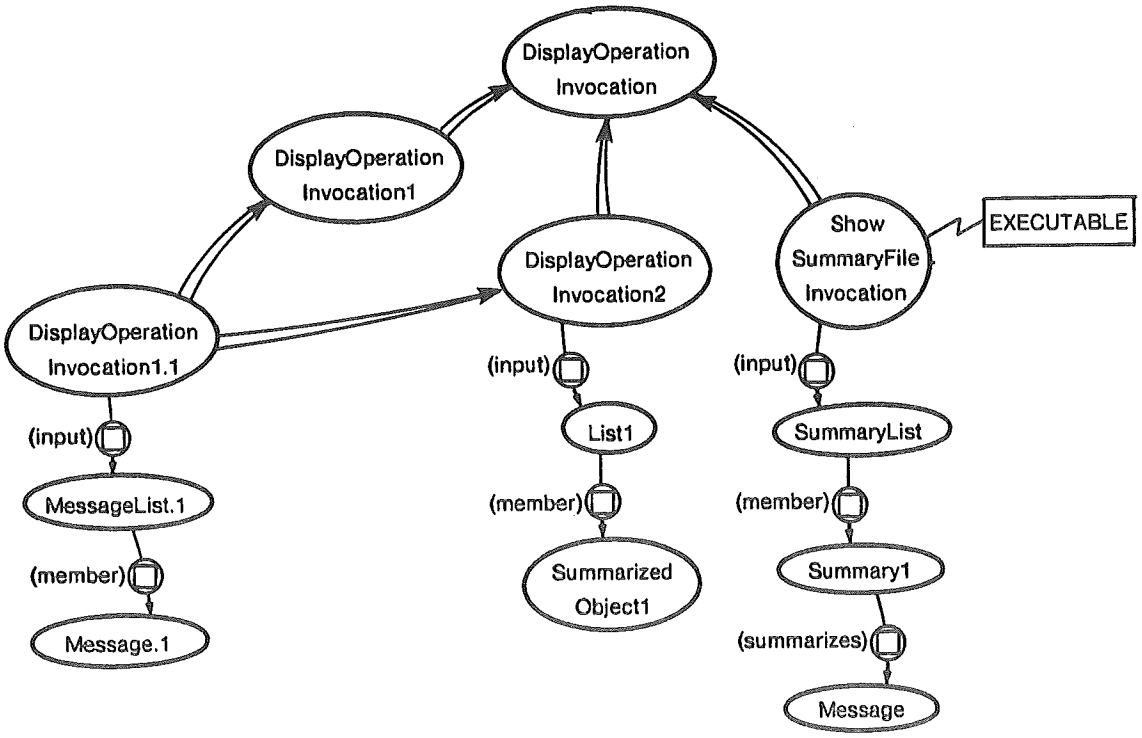


Figure 5-4: Classification in Consul's knowledge base

The rule itself is quite general: users frequently ask to see a list of things (messages, files, etc.) when they really want to see a list of *summaries* of these things (surveys, directories, etc.). Consul's expression of the possibility of transforming the first sort of request into the second is **Rule2**: "if the current description is a display operation to be invoked on a list of summarized objects,[2] then it can be redescribed as a display operation on a list of summaries of those objects."

The application of this rule leads to redescription of the original description as **Display Operation Invocation3.1**. As can be seen in Figure 5-5, this is a specialization of the executable function **Show Summary File Invocation**. The inference process therefore concludes, passing its result description on to the interpreter for execution. If, during execution, the interpreter discovers that the Preconditions of **Show Summary File** (see Figure 5-7) are not satisfied, it will return to the inference process in order to generate a call to open the correct file (in general requiring the application of additional mapping rules). This would finally result in the pair of message system actions mentioned at the beginning of this section and shown in Figure 5-7.

The scenario presented above was rather pat in the sense that there was always a mapping rule (or executable function) applicable to the description at hand. What if this were not the case? For

[2] That is, objects for which summaries exist (e.g., files and their directory entries).

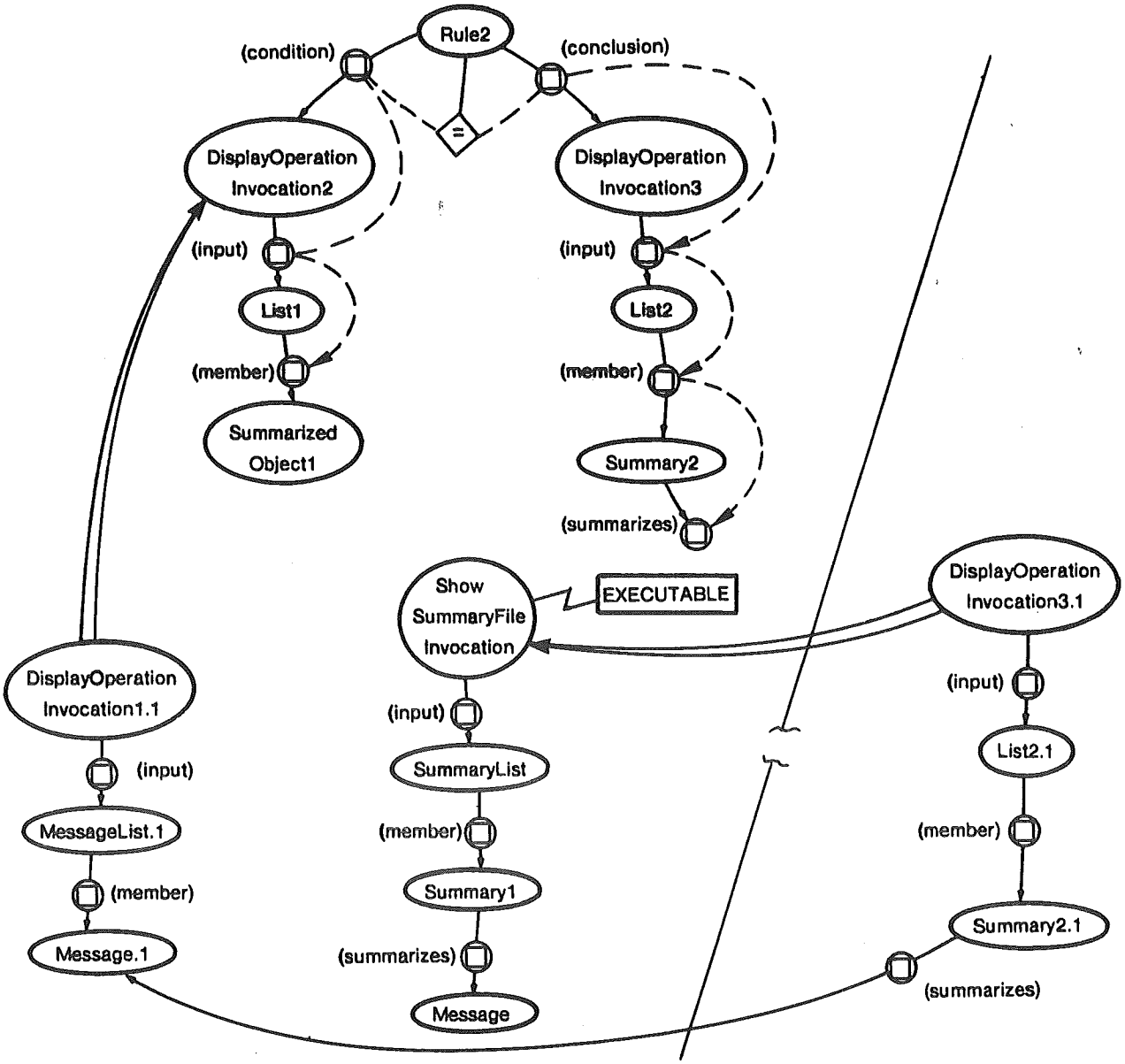


Figure 5-5: Application of Rule 2

example, if in the message service used in the above scenario messages were not always "summarized objects," then **Message.1** would not have specialized **Summarized Object1** in Figure 5-5, and **Rule2** would not have been applicable.

If no applicable function or rule can handle the current description, the system seeks a "closely related" function description to use as a mapping target. The definition of "closely related" is that the two descriptions must share a common ancestor that is a "basic" concept in Consul's actual knowledge base--i.e., the ancestor cannot be part of a rule itself, or a newly generated description. In Figure 5-4, the function description **Show Summary File Invocation** is closely related to the current description **Display Operation Invocation1.1** because they share an appropriate ancestor, **Display Operation Invocation**.

Once a related tool function description is found, it is used as a goal for consequent reasoning. Consul finds the discrepancies between the current description and the desired result--simply the differences that prevent the original description from being classified as a subconcept of the desired description. In this example, **Display Operation Invocation1.1** cannot instantiate **Show Summary File Invocation** because the input of **Display Operation Invocation1.1** is a list of messages (**Message List.1**), while the executable function **Show Summary File Invocation** requires a list of summaries (**Summary List**).

Consul must therefore find a way to redescribe a list of messages as a list of summaries if it hopes to see the current description as an instantiation of this particular executable function. There are two ways to transform the current description: rules and executable functions (since functions produce new descriptions via output and side-effects). Rules are preferable because they save tool execution time; Consul therefore looks for rules first. Consul finds relevant rules by looking in the knowledge base for rule conclusions that produce the "target" part of the discrepancies found earlier. As shown in Figure 5-5, **Rule2** can be found by looking "up" the generalization hierarchy (not all of which is shown) from **Summary List**.

Consul must next be sure that the condition part of the rule is met in the current state of the knowledge base (the current description and the results of previous tool function executions). If the condition is not satisfied, Consul will try to produce the needed state through further rule application and function execution--i.e., through recursive application of the consequent reasoning process.

In this example, in order to satisfy the condition of **Rule2**, there must be a list of summarized objects in the current knowledge base state. This is not currently the case. But suppose that in this message service messages become "summarized objects" after a background process runs. Then the knowledge base would contain a rule that says "if the background process has run since a message was created, then the message can be redescribed as a summarized object." Consul would find this rule during this consequent reasoning process by recursive application of the rule-finding procedure, since it is currently looking for summarized objects. If the background process had in fact run since the messages satisfying the description **Message.1** were created, then the messages could validly be redescribed as summarized objects, and the condition of **Rule2** would be met. **Rule2** could then be applied, and Consul would once again successfully redescribe the initial user request as an executable function.

5.4.3 Making Services Easier To Build: The Process Script Formalism

Consul's ability to provide users with a natural and consistent interface depends on constraining the implementation of interactive services. Part of our research effort has been the formulation of a software methodology to insure the development of services in accordance with the appropriate constraints. Designed to promote the organization of services as collections of small self-describing functional modules, this methodology results in implemented services that are both flexible and "understandable" in terms of knowledge-based inference [9].

As in other software methodologies, Consul's process script formalism is based on the principle of modular design. However, in order to use process scripts, the service builder must take the further step of deciding which modular functions are meaningful to the intended users (e.g., a function to delete a file is likely to be meaningful while one to find a file pointer is not). The decision is, of course, subjective and as such represents a major design consideration of the service builder. In effect, it defines the level of detail to which Consul will be able to understand the functionality of the service.

This understanding is achieved through functional description of the modules via the declarative elements of the process script programming language. Process scripts (see Figure 5-6) give the service builder the means to specify procedures for performing service functions while at the same time describing aspects of the procedures that are important for classifying them in Consul's knowledge base. That is, the description part of a process script enables the procedure to be represented in terms of operations Consul already understands--i.e., that have already been built into Consul's knowledge base. These built-in operation descriptions are important because they have already been related to possible user requests for system action. There are presently seven descriptive categories for process scripts. Additions and modifications to this set may be required as we gain more experience with the formalism.

```
ProcessScript  ForwardOpenMessage;
 Input                      un:UserName;
 Output                     none;
 DataStructuresAccessed     OpenMessage;
 Preconditions              OpenMessageSV = true;
 SideEffects                none;
 Undo                       derived;
 Error Conditions           e1:NoMailBoxForRecipient;
 Body
  begin
   x := FindMailBoxPointer(un);
   if Null(x) then fail with e1;
   AddToInfoField(un);
   y := CreateInfoCitation;
   SendCitation(y,x);
  end;
```

Figure 5-6: A process script

The first descriptor declares the input required by the function. Each input parameter is described by giving it a name and specifying its type. All type definitions are ultimately resolvable into "basic" types already known to the system, thus insuring that the nature of the service-dependent data is known to the system. This type information can then be used in understanding user requests and in answering user questions about the service's functionality (e.g., "What information must I supply to accomplish ...?"). The output descriptor has a similar use. The third descriptor, data structures accessed, identifies other information a service function might use or modify during its execution. For example, a data base that is updated, or variables that are "in common" across functions are described here.

The preconditions descriptor specifies conditions (usually required values of state variables) that must be true before a function can be executed. Preconditions can be used to detect user requests that are incomplete or made in an improper context. State variables must be updated whenever the execution of a service function changes the state monitored by these variables. Consul is informed of these updates via the side effects descriptor. Thus, an inference mechanism, in trying to satisfy the precondition of one process script, could first set up execution of another script whose side effect set the state variable in question. The information specified as side effects can also be used to help answer user questions regarding changes resulting from the execution of a service function (e.g., "What will happen if I do ...?").

The undoability descriptor indicates whether or not the effects of a function can be undone, and if so, how. Knowing which functions are undoable can help Consul decide what to do in response to an ambiguous request. If one possible action is undoable, the system might go ahead and do it, knowing that it can be undone if necessary. On the other hand, a function that could not be undone would never be executed if there were any question as to what the user meant.

The final descriptor is error conditions. Listed here are all the errors that could result during the execution of the process script. Associated with each named error will be some information suitable for presentation to the user.

The body of a process script is just a program (written in a Pascal-like language) that configures smaller functions into more complex ones. Like the "shell" language of UNIX [12], the process script language provides the service builder with an easy way to implement new service functionality in terms of existing functions. Process scripts which have no bodies are called *process atoms* (see Figure 5-7). These are written for service functions whose operations are below the level of user interest or comprehension. The actual code for these process atoms can be written in any programming language and will be executed whenever a process atom is successfully invoked. The description part of the process atom enables the "black box" code it represents to be related to the other operations known to Consul.

```
ProcessScript  OpenSummaryFile;
  Input                      sfn:SummaryFileName;
  Output                     num:EntryNumber;
  DataStructuresAccessed     none;
  Preconditions              none;
  SideEffects                OpenSummaryFileSV := true;
                             CurrentEntryNumber := num;
  Undo                       CloseSummaryFile(sfn);
  Error Conditions           NoAccessRightsToSummaryFile,
                             SummaryFileNotFound;
    Call OpenSummaryFileAtom;

ProcessScript  ShowSummaryFile;
  Input                      none;
  Output                     none;
  DataStructuresAccessed     OpenSummaryFile;
  Preconditions              OpenSummaryFileSV = true;
  SideEffects                none;
  Undo                       ClearScreen;
  Error Conditions           SummaryFileNotDisplayable;
    Call ShowSummaryFileAtom;
```

Figure 5-7: Process atoms

It is important to note that the process script formalism not only gives the service builder the means of functionally describing his system, but also provides him with an interpretively executable programming language. This greatly enhances the system's ability to help the user, since an interpretive environment makes information regarding the dynamic state of the execution readily

available. Although interpretation affects runtime efficiency, the fact that the lower level process atom code of the system will be run in compiled form will make it possible to maintain reasonable performance. The resulting system, executing partially interpretively, strikes a balance between efficient, compiled systems whose execution cannot be analyzed at runtime and totally interpretive systems, which are generally too slow for production environments. More importantly, the portion of the system executed interpretively is exactly that whose effects the service builder decided were meaningful to the user.

As a test of its adequacy for specifying and describing tool functionality, the process script formalism was used to design an interactive calendar system [13]. Approximately eighty process scripts were written (but have not yet been incorporated into the Consul system). The production of these scripts has shown that the formalism is at least reasonable and sufficient from a programming point of view.

5.4.4 New Research Activities

Our efforts during the next three years will be in the following research areas:

Modelling and Acquisition: We must expand all of Consul's models and incorporate techniques for automatically acquiring service-specific knowledge directly from the implementers of that service.

Mapping: All of Consul's processes involving inference, especially explanation and acquisition, must be extended in order to provide the necessary set of user interface facilities.

Programming: In order to encourage construction of services with the characteristics that allow them to be easily incorporated into the Consul system (i.e., small-grained functionality, declarative descriptive information), we must build a software development environment for process scripts.

Execution: Process script execution affects and is affected by the Consul knowledge base; Consul must therefore maintain a process script execution environment to preserve the effects of execution and to allow active communication with knowledge base access and update processes.

5.4.5 Consul's Impact on Future Interactive Systems

The Consul system is a prototype for a class of systems that will have major impact on the interactive systems of the future. Its methodology defines a natural interface to be shared across all services in the system and provides a program development environment that aids in the construction of services to take advantage of that interface.

The effects of having a Consul-like system fall into three major categories:

- the system allows natural interaction with a variety of users,
- changes in services and in the system as a whole become evolutionary rather than disruptive,
- services become easier to build.

Consul's natural communication environment and knowledge-based inference facility allow the system to adapt to different user input characterizations depending on the user's experience with a

service. All of Consul's support will be applied across the full spectrum of services, giving a powerful and uniform interface. Natural language processing allows the user expressiveness not found in command language systems. In future systems, even a single service will often have to deal with a diverse user community ranging from beginners and occasional users to highly experienced users. Consul's parser and inference mechanism are designed to handle both full English sentences and more succinct, stylized forms of expression. This is possible because of the system's model of the underlying semantics of *all* requests, based on its knowledge of user needs and system functional capabilities. That is, it is possible to infer the user's intent from a wide range of input forms expressing that intent. Also, Consul's knowledge base serves as a repository from which to provide help, explanation, and documentation about a service's static characteristics and dynamic behavior.

Change in interactive systems is inevitable. Modification of existing services (bug-fixing, new features) and the introduction of new services are constant, expected activities in the interactive computing world. Consul provides an environment in which both kinds of change can take place with no disruption of the system's interface with the user. The combination of a natural language interface and a mapper insulates the user from the peculiarities of a service's design. The system retains uniformity and consistency from the user's viewpoint. Consul's process script formalism maintains a separation between rapidly changing functional elements and the stable knowledge base that drives the user interface. Consul's acquisition mechanism ensures that changes to services are accounted for in terms of that knowledge base. As a service builder defines a process script for a service, Consul tries to classify it in the existing system knowledge base. Once classified, the script inherits all of Consul's knowledge about the kind of operation it specializes. New functions added in this way will fit a user's evolving model of interactive services.

Consul can also find gaps in the functional coverage of a service. For example, Consul's knowledge base knows that files, in general, can be copied, deleted, undeleted, expunged, etc. If a service builder introduces a data type that is a specific instance of a file, Consul can suggest the need for process scripts that accomplish the full range of operations possible for files. Thus, Consul has a mechanism for helping the service builder provide functional consistency and completeness.

While it is a significant effort to integrate a service into Consul, it is much more difficult to build a similar service without Consul support. The built-in knowledge base and its associated modelling aids make it practical to add domain-dependent information without a special (and very large) basic knowledge representation effort. The design and implementation environment presents a methodology for constructing services that separates interface concerns from low-level functional concerns, provides a framework for modularizing the service, offers uniform error-handling and input/output mechanisms, and checks the implementation against a high-level model of what interactive services should be like. Thus, the service builder finds that part of his work has been done for him, and that the rest of it can be accomplished within a consistent program development environment.

Traditionally, services are self-contained programs that perform a variety of functions. In Consul, a service is a collection of process scripts. Since the atoms (or scripts) of a "service" have been described independently, a service builder may find ways of incorporating those functions into a new service. Of course, the service builder must know how the process scripts he is borrowing work, what conditions must be met before they can be used, what side-effects they may have, and so on. In Consul, that information is explicitly present in the knowledge base.

The Consul system is intended to meet the needs of interactive services in future computing environments. Our design reflects the goal of unifying these new services within a framework offering cooperative interface facilities to both the users and builders of these services.

5.5 SUMMARY OF PROGRESS TO DATE

- The PSI-KLONE parser [2] has been integrated into the system and can handle a variety of sentences.
- The SIGMA message service [16] has been adapted to run under Consul.
- A model of the system, incorporating both service-independent and SIGMA-dependent elements, has been built (without any automatic aids) and presented in [11].
- A mapping mechanism has been built to handle the sentence and rules described in section 5.4.2, and has been reported in [10].
- The process script programming formalism and software development methodology (reported in [9]) has been designed and has been used to implement a number of actual process script programs.
- A rudimentary process script interpreter has been built.

In summary, our research efforts so far have resulted in a workable design for the Consul system; implementation of the parsing, inference, and execution components at a prototype level; a fairly significant knowledge base derived from a good deal of modelling experience; and a demonstration Consul system consisting of the integration of all of these parts. Our research plan for the next three years involves expansion of all components within the current design, and new thrusts in the areas of knowledge acquisition, explanation, and interface architecture.

REFERENCES

1. Bobrow, Robert, and Bonnie Webber, "PSI-KLONE: Parsing and semantic interpretation in the BBN natural language understanding system," in *Proceedings of the 1980 Conference of the Canadian Society for Computational Studies of Intelligence*, CSCSI/SCEIO, May 1980.

2. Bobrow, Robert, and Bonnie Webber, "Knowledge representation for syntactic/semantic processing," in *Proceedings of the National Conference on Artificial Intelligence*, AAAI, August 1980.

3. Brachman, Ronald, *A Structural Paradigm for Representing Knowledge*, Bolt Beranek and Newman, Inc., Technical Report, 1978.

4. Brown, John Seely, and Richard Burton, "Multiple representations of knowledge for tutorial reasoning," in Daniel Bobrow and Allen Collins (ed.), *Representation and Understanding*, Academic Press, 1975.

5. Carbonell, Jaime, and Allan Collins, "Natural semantics in artificial intelligence," in *Proceedings of the Third International Joint Conference on Artificial Intelligence*, IJCAI, 1973.

6. Codd, Edward, et al., *Rendezvous Version 1: An Experimental English Language Query Formulation System for the Casual User of Relational Data Base Systems*, IBM San Jose, Technical Report, 1978.

7. Hendrix, Gary, et al., "Developing a natural language interface to complex data," *ACM Transactions on Database Systems* 3, (2), 1978, 105-147.

8. Kernighan, B. W., and J. Mashey, "The UNIX programming environment," *Software-Practice and Experience* 9, 1979, 1-15.

9 Lingard, Robert W., "A software methodology for building interactive tools," in *Proceedings of the Fifth International Conference on Software Engineering*, 1981.

10. Mark, William, "Rule-based inference in large knowledge bases," in *Proceedings of the National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, August 1980.

11. Mark, William, "Use of database organization in the Consul system," *Joint SIGART/SIGPLAN/SIGMOD (ACM) Publication*, February 1981.

12. Mashey, J. R., "Using a command language as a high-level programming language," in *Proceedings of the Second International Conference on Software Engineering*, pp. 169-176, 1976.

13. Irene Persson, Consul Note 8: "Process Script Definition of a Calendar System," 1980. ISI.

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

15. Schwartz, Jules I., "Construction of software: Problems and practicalities," in Ellis Horowitz (ed.), *Practical Strategies for Developing Large Software Systems*, Addison Wesley, 1975.

16. 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, May 1979.

6. COMMAND AND CONTROL GRAPHICS

Research Staff:
Richard Bisbey II
Benjamin Britt
Dennis Hollingworth
Gertrud Mellstrom
Pamela Norton
Richard Shiffman

Consultant:
Danny Cohen

Support Staff:
Victor Brown
Linda Sato

6.1 PROBLEM BEING SOLVED

As more C2-related information is maintained in computer-based form, computers will need to take more active roles in presenting that data. Computer-generated graphics must supplant volumes of batch-generated printer listings to allow the military to deal with the large quantities of information available and make timely decisions. For example, online computer decision aids will be needed to calculate and graphically display the potential outcomes of alternative strategies. Such a capability must be available in normal, as well as crisis, mode, with particular attention paid to mobility and survivability in crisis mode.

To meet future military needs, command and control graphics systems of the future will need to have the following attributes:

- They must be adaptable to the computation and communications resources that are available. 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, require that the graphics system be tailorable to a wide variety of processor/communications combinations.
- They must be flexible to adapt to and utilize different types of display devices. A wide variety of display devices will be available, ranging from large screen color displays in command centers to more modest hand-held displays in the field. A graphics system must accommodate all of these device types and be capable of displaying the graphics output of any available application program.
- They must be usable in a transnetwork environment. Command centers, host computers, and users will be interconnected by digital networks that will include terrestrial, radio, and satellite communications. Graphics systems for command and control must be designed to operate in this environment.
- They must support the creation of pictures for use outside the immediate application environment. Graphics systems for command and control must provide the capability for storing an application-generated graphics picture in a "graphics file," transmitting the file over a network, and incorporating that picture in another application program, all in a display-device-independent fashion.

6.2 APPROACH

To satisy the above requirements, ISI has developed a distributable architecture for graphics [1]. The architecture allows a graphics system to be constructed as a series of isolatable functions that are pairwise connected by any available intraprocess/interprocessor communications mechanism including telephone, radio, or digital network/internetwork links. Information is communicated between functions using a uniform protocol. The architecture permits a wide variety of configurations ranging from clustering all functions on a single host to distributing each to a different host. Included within the architecture are functional modules that allow the creation of sequential files containing display-device-independent graphics and the incorporation of such files into an existing application's graphics.

ISI has also developed a set of generic graphics primitives (Graphics Language [2]) by which pictures can be described and interacted with at the application level. The particular graphics model used for the language was based on structuring pictures as sets of subpictures which are absolute-transformed segments, as defined by Newman and Sproull [3]. The graphics primitives are transformed by the Graphics System at program execution time into specific operations and display modes appropriate to the device to which the system is connected.

6.3 PROGRESS

Graphics System

In September 1976, Graphics Language was defined. ISI delivered a nondistributed version (1.0) of the graphics system in January 1977, and a distributed version (2.0) in June 1979.

During this reporting period, implementation of all remaining architectural features for the Level 2 distributed system was completed. A display-device-independent graphics file format was defined along with Graphics Language commands for writing segments to or reading segments from externally stored graphics files. Major functional units were added to the graphics system to perform the graphics file reading/writing operations and the necessary transformations between internal protocol and external file representations. The other major architectural feature completed was the automated connection of functional units when the graphics system was used in a distributed, multiple-host configuration. A graphics connection protocol was defined for the ARPANET and network server programs were written. ARPANET connection functions were then added to the graphics system. Also during this period, the graphics system was expanded to include support for Interlisp-10 and Pascal languages, and Hewlett Packard HP-9872A and Advanced Electronic Design 512 display devices. Two reports were published describing Graphics Language [2] and the Graphics System architecture [1].

Other Applications

This period also saw the completion of work on Situation Display. Situation Display is a cooperative effort between ISI and several other DARPA contractors to produce a natural language information retrieval system for distributed databases producing graphics responses. ISI has been responsible for the graphics display portion of the system. During this period, ISI added a graphics file output capability to Situation Display allowing displayed pictures to be saved on disk for later review. Situation Display was also extended to optionally display shipping-lanes on a geographic plot, as well as to allow a tabular display of force data.

Finally, a general-purpose graphics file display program was written. The program is menu-driven and provides a friendly interface allowing a novice user to configure the graphics system and sequentially display graphics files.

6.4 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, storage, 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.

6.5 FUTURE

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

- Design of a highly portable graphics system "back-end" capability.
- Design and development of a computer-based briefing aid capability.

Highly Portable Back-end

To date, the Graphics System has been implemented on PDP-10s and 20s operating the TENEX and TOPS-20 operating systems, and PDP-11/70 Remote Site Modules operating the UNIX operating system, all of which are large and expensive mainframe computers. An analysis of the Graphics System shows that its performance is largely limited by the channel bandwidth between the last Graphics System function, the Display Order Generator (DOG), and the display device. Users who do not have a high bandwidth channel between the display device and the host on which the DOG function runs (such as those connected via TTY or ARPANET links) obtain very poor display performance.

With the recent availability of very powerful, very small 16-bit microprocessors such as the Motorola MC68000, it is both practical and desirable to migrate Graphics System functions to such a processor. Two major benefits accrue. First is the small physical size. Unlike a PDP-10 or PDP-11 computer that occupies one or more equipment racks, a microprocessor-based system may occupy one or two 16x9 inch cards, small enough to be packaged with the display device. This allows portability of operation while retaining maximum graphics display performance. A second major benefit is cost savings. A microprocessor-based system can provide the same functional capability of a mainframe host at a fraction of the cost.

Briefing Aid

Based on experiments using our graphics file display utility program, ISI will develop a Briefing Aid application program. The Briefing Aid will permit the creation and presentation of briefings using computer graphics rather than conventional film media. The online graphics output of any command and control application program, such as Situation Display, will be readily incorporated into a briefing. Briefing slides will be displayed sequentially under the control of a separately prepared briefing script

or will be displayed individually under direct briefer control. The distributability and display-device-independent properties of the Graphics System will permit briefings to be given anywhere a display terminal can be located, such as in a commander's office or at a mobile headquarters.

REFERENCES

1. Bisbey, R., II, and D. Hollingworth, *A Distributable, Display-Device Independent Vector Graphics System for Command and Control*, USC/Information Sciences Institute, RR-80-87, 1980.

2. Bisbey, R., II, D. Hollingworth, and B. Britt, *Graphics Language*, USC/Information Sciences Institute, TM-80-18, 1980.

3. Newman, W. M., and R. F. Sproull, *Principles of Interactive Computer Graphics*, second edition, McGraw-Hill, 1979.

7. INTERNETWORK CONCEPTS RESEARCH

Research Staff:
Jon Postel
Danny Cohen
Carl Sunshine
Bernard Berthomieu
Suzanne Sluizer
Greg Finn

Research Assistants:
Alan Katz
Paul Mockapetris

Support Staff:
Linda Sato

7.1 PROBLEM BEING SOLVED

This project explores the design and analysis of computer-to-computer communication protocols in multinetwork systems. The project has three task areas: (1) Analysis, (2) Applications, and (3) Design and 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 and Concepts is concerned with the development of network and transport protocols, in particular the Internet Protocol (IP) and TCP, and seeks new approaches in the application of packet switching to communication problems.

7.2 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. The designs will be 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. These protocols will have to be 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.

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

Computer mail is the application we will use as a focus for demonstrating internetwork service. Within this application area we are developing a multinetwork, multimedia mail service. Our interest is primarily in the communication mechanisms, rather than the user interfaces. We are developing a set of procedures and data structures to be used in multimedia mail, and have embodied these in a message processing module.

We have assisted in the design of several protocols in the internet family. The areas of addressing, routing, and multiplexing are particularly subtle and require careful attention. We have focused attention on these areas and have explored many options in technical discussions and memos. Our

approach is to develop an understanding of these technical issues and to advocate the inclusion of general-purpose supporting mechanisms in the protocol specifications.

7.3 SCIENTIFIC PROGRESS

Protocol Analysis

Our work this year has been divided into two major areas: a wide survey and evaluation of potential methods for formal specification and verification of protocols, and a deeper experimentation with one particular approach.

In the first area, we have completed an extensive survey of the literature, and experimented briefly with the methods found. The survey and experimentation have allowed us to understand the relationship between different methods, classify them into appropriate categories, and evaluate their relative merits. Specification categories include finite state automata, abstract machines, formal languages, sequencing expressions, Petri nets, buffer histories, abstract data types, programs, and temporal logic. Verification methods include state exploration, symbolic execution, structural induction, inductive assertions, and design rules.

Abstract machines appear to be a promising specification method because of their understandability and the availability of automated analysis tools. However, the need to invent and manipulate an explicit state may be viewed as a disadvantage, leading to consideration of sequencing expression or buffer history methods which avoid explicit state notions. Other difficulties in treating progress and liveness properties are addressed by temporal logic methods.

The results of this survey are presented in [18] and [38].

Our in-depth experiments have focused on combining the understandability of abstract machine specifications with the analytical power of an automated abstract data type system called ***Affirm*** developed by the ISI Program Verification project (see their writeup in this report, chapter 2). A variety of protocols have been treated, from the simple "alternating bit" protocol [13,35] and "selective repeat" protocol [36], to a more sophisticated window-based data transfer protocol [24], and the "three-way handshake" connection establishment protocol (used in TCP). Finally, a simple transport protocol service including data transfer, flow control, connection establishment, and multiplexing capabilities has been specified [14].

The formal specification of these protocols has been a significant accomplishment in its own right, and has forced us to clarify a number of ambiguous features about the protocols. In addition, we have developed verification methods for determining the properties of a single protocol specification and for demonstrating the consistency between a protocol specification and a completely independent service specification (i.e., showing that the protocol provides the intended service). This verification work makes heavy use of the theory and automated tools in the ***Affirm*** system and is still under way.

Liveness properties present special difficulties for abstract data type models, and we have obtained some interesting results by generalizing decreasing measure function techniques to show eventual termination and using preconditions to show deadlock-freeness [35,36].

Protocol Applications

We completed the design for an internet multimedia message system. Separate specifications define the overall structure and control format [3,26], and the body structure and format [27,30]. The focus in this design is the communication of messages in a machine-oriented internal representation which provides for carrying data of several media including text, voice, facsimile, and graphics. Significant progress was made on the implementation of a prototype message processing module (MPM) following this design.

The Multimedia Mail System

This message system model takes the view that the message service can be divided into two activities: message reading and composition, and message delivery. Reading and composing messages are interactive activities that involve a user interface process (UIP). The message delivery activity may be carried out by background processes, MPMs. Our work concentrates on message delivery and leaves the development of sophisticated user interfaces to other projects (e.g., Cooperative Interactive Systems--see chapter 5).

The internetwork multimedia 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 and that the message delivery protocol is implemented in an MPM process. The MPMs exchange messages by establishing full duplex communication and sending the messages in a tightly specified format. The MPMs may also communicate other information by means of commands.

A user writes a message by interacting with a UIP. He may use 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, he sends it by placing it in a data structure shared with the MPM.

The MPM takes the data, adds control information to it, and transmits it. The destination may be a mailbox on the same host, a mailbox on another host in the same network, or a mailbox in another network. 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 typically provides calls to open and close connections and to send and receive data on a connection. The MPM receives input and produces output through data structures that are produced and consumed respectively by UIPs or other programs. The MPM transmits messages, including control information, in a highly structured format using typed data elements in a machine-oriented yet machine-independent data language.

Our work on this system this year included the completion of a draft MPM implementation which resulted in many small changes in the original design and specification. The second specification was completed [26], and the second MPM implementation was begun.

The Interim Mail System

In this mail system the goal is simply to bridge the gap between the different interprocess communication systems presented by the Network Control Protocol (NCP) and the TCP host-to-host protocols. Our approach is to create a mail relay process on a host that has implemented both NCP and TCP [32]. This mail relay process will accept mail from either NCP or TCP sources and relay it to

either NCP or TCP destinations. For the mail relay process to work, the control protocol of the mail procedure must be slightly elaborated to carry additional address information. Work on this task began very late in the year. We did produce a specification of the new procedures [33], and we also began the implementation of the relay process.

Protocol Design and Concepts

We contributed to and edited two editions of the Internet Protocol and the Transmission Control Protocol Specifications [7,8,10,11]. The Internet Protocol is a datagram-style gateway-level protocol that provides the addressing, routing, and fragmentation/reassembly functions in the internetwork. The Transmission Control Protocol is a connection-style host-level protocol that provides end-to-end reliable ordered delivery of streams of data. These specifications [10,11] have been adopted as the basis for a DoD internetwork protocol standard.

In addition we have produced specifications for the higher level protocols and applications: telnet, file transfer, and user datagrams [21,22,28].

We have contributed to general planning of the internet communication system, in particular, the identification and discussion of several issues, including addressing, multiplexing, and routing. Published papers on these topics are [4,5,19,37]. We also wrote Internet Experiment Notes (IENs) and made presentations at Internet Working Group meetings on these topics [2,6,15,16,29].

We participated in the Internet Working Group and provided support for the group's activities by producing meeting agendas and minutes, and other routine reports [1,9,12,20,31,34]. We also coordinate the monthly report for the ARPA Internet Program.

7.4 IMPACT

Protocol Analysis

The use of more precise specification methods will facilitate cheaper, faster, and more reliable implementation of the ever-increasing number of communication protocols in DoD computer networks. The research described here has already had some impact on major protocol development projects sponsored by the Defense Communications Agency and by national and international standards groups. The previously widespread informal narrative specification methods are being augmented by more formal specifications, particularly of state transition or abstract machine variety.

The development of protocol verification techniques also promises to improve reliability and reduce debugging time in implementing network systems. By allowing analysis of protocol designs prior to actual implementation, problems are detected earlier. Esoteric bugs that would be difficult to find with ordinary testing and debugging may also be revealed by formal verification. Several such bugs have already been discovered and eliminated from the TCP.

This research program has also had an impact by influencing other government-sponsored research projects, particularly in the area of program verification, toward developing analysis techniques that are applicable to computer networks. In cooperation with the Program Verification project, we hosted a workshop in July 1980 attended by personnel from over a dozen other projects where applicability of general methods to network problems was discussed. Many of these projects have followed up with work on protocols.

Protocol Applications

Computer mail is the most significant use of the new communication capability provided by packet switching networks. Our work to extend the range and capabilities of computer mail will have important consequences for DoD.

The potential for multimedia communication in a computer-assisted environment is great. The ability to communicate diagrams or maps and then to talk about them will increase the effectiveness of remote communication tremendously. The combination of text, speech, graphics, and facsimile into a common framework and data structure may have substantial impact on other applications as well.

The power of a communication system is directly related to the number of potential communicants. For computer mail, this means that the power of a system is related to the number of people who have access to that system. To have access to a computer mail system requires the use of compatible components: terminals, programs, and protocols. Our work on protocols and programs will increase the power of computer mail by enlarging the set of compatible components.

Protocol Design and Concepts

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 ARPA community's work on DoD communication systems.

Through our participation in discussions at the Internet Working Group meetings, and in technical meeting with other contractors we have successfully influenced the development of many protocols and protocol features.

7.5 FUTURE WORK

Protocol Analysis

We plan to conclude our work in protocol verification in the next 18 months. We expect to report on the experience with the verification systems used and the result of attempting to verify TCP.

Protocol Applications

We will continue our experimental development of an internet multimedia system. In the next year we will begin to experiment with the actual entry and delivery of multimedia messages.

We will continue our work on the text mail relay system, which we expect to complete within the next year.

Protocol Design and Concepts

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

We will continue to raise important issues for discussion in the Internet Working Group and to seek out and develop new opportunities to utilize the capabilities of packet-switched communication systems.

REFERENCES

1. Postel, J., Internet meeting notes - 10, 11, 12, and 13, September 1979, IEN-121, USC/Information Sciences Institute, October 1979.

2. Cohen, D., On addressing and related issues (or, Fuel for a discussion), USC/Information Sciences Institute, IEN-122, October 1979.

3. Postel, J., "An internetwork message structure," *Sixth Data Communication Symposium*, ACM/IEEE, November 1979, pp. 1-7.

4. Cohen, D., and J. Postel, "On protocol multiplexing," *Sixth Data Communication Symposium*, ACM/IEEE, November 1979, pp. 75-81.

5. DiCiccio, V., J. Field, E. Manning, and C. Sunshine, "Alternatives for the interconnection of public packet switching data networks," *Sixth Data Communication Symposium*, ACM/IEEE, November 1979, pp. 120-125.

6. Cohen, D., Summary of the ARPA/ETHERNET community meeting, USC/Information Sciences Institute, IEN-126, November 1979.

7. Postel, J., DoD standard Internet Protocol, USC/Information Sciences Institute, IEN-123, December 1979.

8. Postel, J., DoD standard Transmission Protocol, USC/Information Sciences Institute, IEN-124, December 1979.

9. Postel, J., Assigned numbers, USC/Information Sciences Institute, IEN-127, RFC 762, January 1980.

10. Postel, J., "DOD standard Internet Protocol," IEN 128, RFC 760, USC/Information Sciences Institute, NTIS ADA079730, January 1980. Appears in *Computer Communication Review, Special Interest Group on Data Communications*, ACM, 10 (4), October 1980, 12-51.

11. Postel, J., "DOD standard Transmission Control Protocol," IEN 129, RFC 761, USC/Information Sciences Institute, NTIS ADA082609, January 1980. Appears in *Computer Communication Review, Special Interest Group on Data Communications*, ACM, 10 (4), October 1980, 52-132.

12. Postel, J., Internet meeting notes - 4, 5, and 6 February 1980, USC/Information Sciences Institute, IEN-134, February 1980.

13. Sunshine, C., Axioms for the Alternating Bit Protocol, USC/Information Sciences Institute, ***Affirm*** Memo-17-CAS, February 1980.

14. Schwabe, D., Transport protocol service specification, ***Affirm*** Memo 19, March 1980.

15. Sunshine, C., Addressing mobile hosts in the ARPA internet environment, USC/Information Sciences Institute, IEN-135, March 1980.

16. Cohen, D., On holy wars and a plea for peace, IEN-137, USC/Information Sciences Institute, April 1980.

17. Postel, J., Time server, USC/Information Sciences Institute, IEN-142, April 1980.

18. Bochmann, G., and C. Sunshine, "Formal methods in communication protocol design," *IEEE Transactions on Communication*, COM-28, 4, April 1980.

19. Postel, J., "Internet protocol approaches," *IEEE Transactions on Communication*, COM-28, 4, April 1980.

20. Postel, J., Internet meeting notes - 14 and 15 May 1980, USC/Information Sciences Institute, IEN-145, May 1980.

21. Postel, J., Telnet protocol specification, USC/Information Sciences Institute, IEN-148, RFC 764, June 1980.

22. Postel, J., File transfer protocol specification, USC/Information Sciences Institute, IEN-149, RFC 765, June 1980.

23. Plummer, W., and J. Postel, TCP JSYS calling sequences, Bolt Beranek and Newman, IEN-150, June 1980.

24. Thompson, D., A behavioral axiomatization of the Stenning data transfer protocol, ***Affirm*** Memo 16, USC/Information Sciences Institute, June 1980.

25. Sunshine, C., Problem areas in protocol specification and verification, ISI internal memo, July 1980.

26. Postel, J., Internet message protocol, USC/Information Sciences Institute, IEN-113, RFC 759, August 1980.

27. Postel, J., A structured format for transmission of multimedia documents, USC/Information Sciences Institute, RFC 767, August 1980.

28. Postel, J., User datagram protocol, USC/Information Sciences Institute, RFC 768, August 1980.

29. Cohen, D., Controlled routing in the Catenet environment, USC/Information Sciences Institute, IEN-156, September 1980.

30. Postel, J., Rapicom 450 facsimile file format, USC/Information Sciences Institute, RFC 769, September 1980.

31. Postel, J., Assigned numbers, USC/Information Sciences Institute, RFC 770, September 1980.

32. Cerf, V., and J. Postel, Mail transition plan, Advanced Research Projects Agency, RFC 771, September 1980.

33. Sluizer, S., and J. Postel, Mail transfer protocol, USC/Information Sciences Institute, RFC 772, September 1980.

34. Postel, J., Internet protocol handbook table of contents, USC/Information Sciences Institute, RFC 774, October 1980.

35 Berthomieu, B., Proving progress properties of communication protocols in ***Affirm***, USC/Information Sciences Institute, ***Affirm*** Memo-35-BB, October 1980.

36. Berthomieu, B., Selective repeat protocol: Axiomatization and proofs, USC/Information Sciences Institute, ***Affirm*** Memo-36-BB, October 1980.

37. Sunshine, C., "Current trends in computer network interconnection," *Advances in Data Communication Management*, T.A. Rullo (ed.) Heyden, 1980.

38. Sunshine, C., "Formal modeling of communication protocols," in *Proceedings of the Conference on Communication in Distributed Data Processing Systems*, Technical University Berlin, January 1981. Also USC/Information Sciences Institute RR-81-89, March 1981.

8. NETWORK SECURE COMMUNICATION/ WIDEBAND COMMUNICATION

Research Staff:
Stephen Casner
William Brackenridge
Danny Cohen
E. Randolph Cole
Eric Mader

Support Staff:
Jeff LaCoss
Gertrud Mellstrom
Robert Parker
Linda Sato
Jerry Wills

8.1 INTRODUCTION

The ISI Network Secure Communication (NSC) project has been instrumental in the development of protocols and real-time systems to transmit packetized voice over the ARPANET, both in point-to-point conversations and multisite conferences. The project is now broadening its scope as the Wideband Communication (WBC) project, which will develop the technology required for the future support of thousands of simultaneous conversations being transmitted over a wideband satellite channel in the internetwork environment. It will advance packet voice from a demonstration program to an experimental system continuously available for use in the transaction of normal daily business.

While the NSC project concentrated on voice communication, the WBC project will work on integrated communication of several media, including voice. The goal is to develop real-time multimedia teleconferencing using wideband packet-switched networks. In addition to voice, the initial emphasis will be on the development of a video bandwidth compression system which operates in real time and takes advantage of the ability of a packet-switched network to accommodate varying bandwidth requirements.

8.2 PROBLEM BEING SOLVED

The military communicates vast amounts of information in several forms, including voice, text messages, and graphs. To a large extent, these various forms are not integrated; in fact, frequently they are communicated over independent networks. To improve the survivability of critical communication, integration is necessary to allow any part of the available communication bandwidth to be used for any of the types of data. Since the mix of data varies dynamically, only a fully integrated communication system can ensure that the most essential information can always be communicated.

A good way to integrate the communication of voice, data, and other media is with a packet switched network. A study by the Network Analysis Corporation [2] showed that packet switching technology with an integrated voice/data network is the most economical way to meet DoD communication needs, as compared to circuit switching, fast circuit switching, and hybrid switching over a wide range of parameters. A further advantage of digital packet switching for voice communication is that it can be secured to any degree desired, unlike analog voice communication.

The first purpose of the Wideband Communication Program is to find the effects of scale on a packet voice communication system. The ARPA NSC effort has demonstrated that it is feasible to transmit real-time, secure speech through a packet network. However, this was done using minicomputers with attached high-speed signal processors; such systems are flexible but relatively expensive, so only a few exist. Advances in VLSI (very large-scale integration) technology, combined with sufficient experience in packet speech protocols and software, now make it possible to create a more economical voice terminal. Current ARPA plans call for the production of 10 to 100 such devices.

Likewise, the bandwidth of previously available packet networks has limited the number of simultaneous voice conversations to just a few. With the advent of the wideband satellite network, it will now be possible to simultaneously transmit about 20 uncompressed voice streams or perhaps 1000 narrowband voice streams. Therefore, it is now time to test packet speech in an environment which more closely resembles future real-world applications, an environment in which the packet-switching technology can take advantage of the bursty nature of speech to demonstrate its efficiency in statistical multiplexing.

If packet switching is to meet the total DoD communication requirement, then it must also expand to support media which have not been accommodated in the past. A prime example is real-time, full-motion video. The large bandwidth required by the video signal will make it as difficult to transmit video across the wideband satellite network as it was to transmit voice across the lower bandwidth ARPANET. This will again restrict communication to one or two channels, but it is expected that packet-switched networks will continue to grow in capacity so that multiple channels of video will be be feasible in the not-too-distant future.

The switch to satellite communication introduces some new problems that must be investigated, specifically, increased delay and higher error rates. New transmission protocols must accommodate these problems and at the same time take advantage of added capabilities, such as broadcast transmission.

8.3 GOALS AND APPROACH

The ISI Wideband Communication project has two primary goals:

1. To develop the technology required to support future packet speech systems with thousands of voice channels, beginning with an experimental facility accessible by a much broader user community than that of the first demonstration systems.
2. To explore new modes of packet communication made possible by the increase in bandwidth of the satellite network over that of previous packet networks, and to investigate how the added bandwidth could benefit old modes of communication.

To work toward these goals, the WBC project's efforts cover several areas. Specifically, the project is working to:

- Continue the design and development of the protocols required to support communication in the Wideband Network as part of the larger ARPA internet environment. Included are the protocol extensions required to support multimedia conferencing with voice, video, graphics, and other data.
- Interface the Wideband Network to the Switched Telephone Network (STN) to promote the widespread use of packet voice. The WBC project has designed and developed an

STN interface board which will be distributed to the Wideband Network sites. ISI and others will integrate the STN interface with voice terminal equipment to make the Wideband Network accessible to more users on a continuous experimental basis.

- Design and implement the hardware and software required for the transmission of packetized narrowband video over the satellite. The goal will be to build a system capable of real-time transmission of color video with moderate motion.
- Investigate the performance of the satellite network by itself and in combination with terrestrial networks to determine how protocols should be tuned for various network conditions and types of data.

The STN interface will expand the packet speech user community by providing access via the telephone network using dial-in and dial-out facilities. The interface utilizes a wideband speech vocoder because it functions well with telephone-quality input and its cost is significantly lower than that of narrowband vocoders. The low cost will allow a reasonable number of vocoders to be supplied, and telephone access will enable more people to use them. This system will allow packet speech to be used as an on-going experiment by a large number of people at the participating sites; it will provide experience with a more heavily loaded system than one used primarily for demonstrations.

The STN board has been designed to interface directly with the Lincoln Laboratory Voice Terminal because several of the latter will be fabricated and distributed to the Wideband Network sites. ISI will likewise fabricate and distribute STN interfaces so that each site can set up telephone access facilities, allowing the Wideband Network to be tested with routine cross-country telephone calls.

The objective of the WBC project's video experiments is to investigate and demonstrate transmission of video data over a packet-switched network for the first time. The ultimate goal will be to build a system capable of transmitting color video with moderate motion in real time at a data rate of 1.5 megabits per second. The word "video" means television camera images at frame rates ranging from one frame or less per second (frame-grabbing) to 30 frames per second (full motion). Monochromatic video will be used initially, with color to be used later.

The Wideband Network provides an opportunity to utilize novel video bandwidth compression techniques which have been developed but not utilized because they are not well matched to fixed-rate transmission channels. Some of the best video bandwidth compression algorithms available today generate compressed data at a variable rate. A packet-switched channel allows the statistical multiplexing of several variable rate streams to use the available bandwidth more efficiently.

The approach for video bandwidth compression algorithms will be to evaluate present algorithms and adapt them to the wideband channel, not necessarily to develop new ones. The object is to select the best current algorithm or family of similar algorithms, shape it to best fit the wideband channel, and implement it to run in real time.

In addition to the voice and video experiments described above, the WBC project will investigate the transmission of nonreal-time data, such as large files, on the Wideband Network. Such data will be sent in datagram (contention) packets which fill in the bandwidth unused by the streams reserved for voice and video traffic. It is the use of one hybrid network for both kinds of data which provides the economy expected for the packet network.

To gain the full benefit of the large bandwidth of the satellite network, the file transfer protocol parameters will need to be tuned to fit the performance of the Wideband Network and the other

networks involved in accessing it. For example, if the Internet Transmission Control Protocol (TCP) is used, it will require a huge window to allow several packets to be outstanding at a time. Another modification would be to return negative acknowledgments (NAKs) for packets which arrive with checksum errors in the data but not in the header. Since the Wideband Network will have a relatively high error rate and a long delay, returning NAKs could cut down the amount of time wasted waiting for timeouts to expire. These protocol issues and others will require investigation before the Wideband Network can be used effectively.

8.4 PROGRESS

8.4.1 Voice Authentication

ISI has implemented a real-time version of the text-independent Voice Authentication System (VAS) developed at the Speech Communications Research Laboratory (SCRL) by John Markel [3]. The system was demonstrated by ISI at the NSC contractors' meeting at SRI International in October 1979. The VAS is included as an integral part of the "SPEECH" conferencing system which runs on ISI's PDP11/45 and FPS AP-120B hardware under the EPOS operating system. It functions as an adjunct to the voice bandwidth compression algorithm (Linear Predictive Coding, or LPC).

With the VAS, a conference participant at ISI can authenticate any participant in the conference, both local and remote, using profile vectors compiled during training sessions. Because the system is text-independent, the participant need not be aware that his speech is being collected, either for training or authentication. As currently implemented, the VAS is trained on an unconstrained segment of each user's speech lasting about 10 minutes. This speech is analyzed to form an LPC parameter file, which is then processed by a FORTRAN program to create the profile, which is then stored on a disk.

To authenticate a speaker, the system processes successive segments of 300 voiced LPC frames (about 10 seconds of speech), then compares the statistics for each segment against a profile selected by the user. This results in a measure of the speaker's distance from the profile during each segment. The user is presented with a running tally of yes/no decisions based on a preset threshold for the distance.

The real-time VAS uses quantized LPC parameters, since it must handle speech which has been transmitted across the ARPANET. Although the differences have not been thoroughly measured, its performance seems fairly close to that of the original SCRL version, which uses unquantized parameters. The real-time VAS has given excellent results in tests to date.

8.4.2 NVP-II and ST Protocols

The second-generation Network Voice Protocol (NVP) for the transmission of packet speech has been designed. The voice-specific control and data protocol, NVP-II, is used in conjunction with a new stream transmission protocol, ST, which provides control of reserved-resource streams on those networks which support them. Several parties collaborated in the design of these protocols, but NVP-II was written primarily by ISI, and ST was written primarily by Lincoln Laboratory.

NVP-II is designed to replace the first NVP which has been operational since 1974. Its main improvements over NVP-I are in the following areas:

- Operation in the ARPA internetwork environment.
- Improved conference facilities.
- Provision of the "handles" necessary to achieve high efficiency of intermediate networks and utilization of special features like stream-setup, when necessary, and broadcasting.
- More flexible control to allow future extensions such as the support of multimedia communication.

NVP-II, like NVP-I, is divided into a Control Protocol (CP) and Voice Data Protocol (VP).

The module which implements the control protocol is called the CM. Its main objectives are (i) establishing and monitoring the communication paths, and (ii) supporting a convenient user interface. The control functions are performed by exchanging "control instructions" between the participating CMs. These instructions are referred to as control-tokens.

The voice data protocol is basically as defined for NVP-I, and is not expected to be intrinsically changed. All the forthcoming acoustic improvements of any voice digitization algorithm should apply equally to both protocols. However, NVP-II packages the voice-data in a different way than NVP-I does.

NVP-II supports two main communication modes:

1. Point-to-Point (PTP) full-duplex, two-party communication, and
2. Multi-Destination "Half-Duplex" (MDHD) communication (conference).

NVP-II, unlike NVP-I, is symmetric in the sense that in a two-party call, the distinction between the CALLER and the CALLEE does not exist, and in conference there is no information regarding whether a participant called the conference or vice versa. On the other hand, the supporting communication protocol, ST, is based on asymmetric full-duplex connections with the notion of "direction" (i.e., CALLER and CALLEE) which is maintained throughout the entire life of the connection.

8.4.2.1 Implementation in SPEECH

The WBC project has implemented the ST and NVP-II protocols to support point-to-point voice communication using ISI's SPEECH system running under EPOS on the PDP11/45 and FPS AP-120B hardware. In the current configuration used for protocol debugging, the ARPANET functions directly as a local network accessed by the ST module. In the future, this will be replaced by a connection through a local access network and gateway to the wideband satellite network.

The implementation consists of modules:

1. The ST dispatcher module with an ARPANET interface.
2. The NVP-II control protocol module.
3. The NVP-II data protocol module.

The first two modules are written in the programming language "C" to increase the capability of sharing this code with other implementations. The NVP-II implementation currently handles only point-to-point connections, but in FY81 it will be extended to support multiple-site conferences.

In order to enable the SPEECH system to interoperate with the PCM vocoder in the STN interface, the PCM algorithm has been programmed in the FPS AP120-B signal processor. This PCM will also be used to communicate over the Wideband Network with other voice terminals at ISI and Lincoln Laboratory.

8.4.3 Switched Telephone Network Interface

The WBC project has designed an interface to allow connection of voice terminals to the commercial Switched Telephone Network (STN). The interface will allow the packet speech system to 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) can be provided without additional cost or effort. The interface is built on a 7-in. square circuit board which plugs directly into a Lincoln Laboratory Voice Terminal. A diagram of the interface's functional components is shown in Figure 8-1.

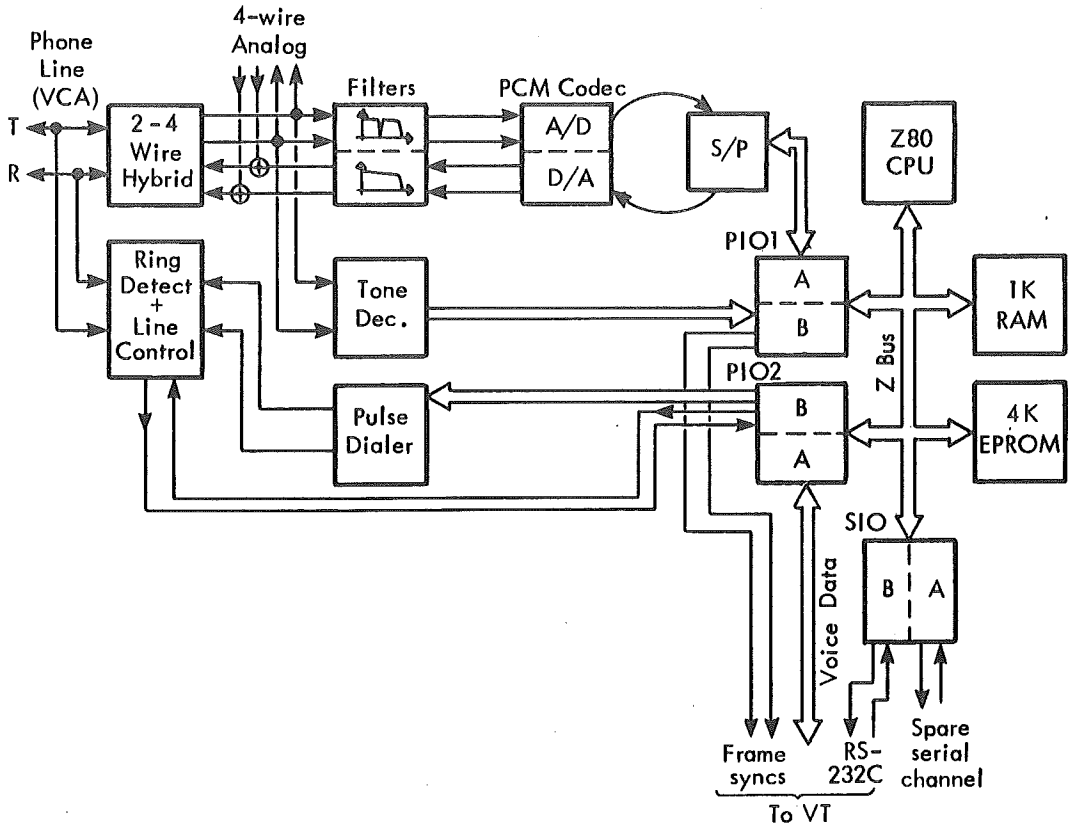


Figure 8-1: ISI STN interface

The STN interface detects touch-tone (DTMF) pulses in the audio from the phone line and translates them into control codes delivered to a voice terminal. This allows a user of the system to specify a connection through the Wideband Network by using the pushbuttons on his phone.

Likewise, the phone line interface translates control codes from the processor into dial pulses 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 audio channel to let the caller decide whether or not the call completed successfully.

The vocoder included in the STN interface is a 64Kb/s PCM codec. It is generally desirable to reduce the bandwidth required for speech communication, but for the satellite network there are several advantages to the use of wideband speech:

- PCM is the standard in the digital telephone industry.
- PCM hardware is simple, low-cost, and readily available.
- It will require fewer channels to place a large load on the network for experimental purposes.
- The high quality of PCM speech will make it easier to isolate network-related glitches from vocoding-related glitches (e.g., pitch tracking errors in LPC).
- PCM can be used in tandem with other vocoders if necessary.

The heavy loading which PCM places on the system is an initial advantage for testing and evaluation; however, as narrowband vocoders improve in quality and become less expensive, and as the satellite system becomes better understood, further experimentation will use a larger number of narrowband vocoders.

The prototype of the STN interface board has been assembled, and an initial test configuration of the firmware for the board's Z80 microprocessor has been completed. The test firmware has been used to verify the operation of the components of the interface, including detecting DTMF signals, pulse dialing, PCM coding/decoding, and silence detection. It was determined that the microprocessor can handle the fairly taxing processing requirements. The firmware will now be extended to the full operational capability.

8.4.3.1 STN Dialing Sequences

The WBC project has begun the definition of a set of dialing sequences to be used when an STN interface is called from a telephone. This definition will specify the way to dial local telephone numbers at remote sites and other aspects of connection control from a pushbutton telephone. NVP-II will be expanded to include these dialing sequences and any special requirements for the interconnection of the STN and the Wideband Network.

8.4.4 EPOS Operating System

The EPOS operating system developed by the ISI NSC project was selected in February 1980 to support the Mini-Concentrator gateway software being developed by Lincoln Laboratory. The Mini-Concentrator will act as a packet aggregator using the ST protocol, and will run on PDP11/44 machines at ISI, Lincoln, and SRI. EPOS has been augmented with some additional capabilities required for the development of the Mini-Concentrator, and several other support tasks have been accomplished:

- An *EPOS System Overview* [1] document was prepared to help the individuals making the selection evaluate EPOS relative to the other operating systems which were considered.

- EPOS has been augmented to fully support the UNIX file system in order to allow Mini-Concentrator development to be performed on the PDP11/45-based UNIX systems which run at Lincoln and ISI. This allows files created by either operating system to be read by the other.
- ISI generated an EPOS system for the Lincoln configuration, and it was transferred to Lincoln via the ARPANET. The system was loaded and ran successfully on the first attempt without requiring any participation from ISI.
- Terminal line flow control (using XON/XOFF control characters) has been added in both directions. This allows EPOS and UNIX running in separate machines to be connected via terminal lines for downloading purposes. Without the flow control, either system might be flooded by character input at high rates.
- The EPOS debugger (MEND) has been updated to accept the revised MEND/XNET symbol table format. A transformation from UNIX symbol format to the MEND format has been defined; a program was written by Lincoln implementing this transformation so that EPOS applications developed on UNIX can be loaded and run on EPOS.
- The WBC project has developed a program called XLDR which runs under EPOS to download software into ISI's second PDP11/45 from the primary one over a terminal line.
- A library of subroutines has been written to allow programs written in C to access the EPOS system calls. The Mini-Concentrator software will be written primarily in C. Several test programs have been written to check out the subroutine library. The library has been supplied to Lincoln Laboratory.
- A second library of subroutines which emulate UNIX system call subroutines has also been written. It will allow some standard UNIX programs to run on EPOS. Several support programs have already been adapted. Included are the disk maintenance programs ICHECK, DCHECK, and NCHECK; the first two have been set up to run automatically when the system is started to verify the filesystem's integrity. Also adapted are two UNIX programs LD and ATOLDA, part of the UNIX software production path.
- The Whitesmiths "C" compiler was converted from an RT11-based program to an EPOS program so it could be used for the development of the NVP-II software being added to the SPEECH system. The EPOS system calls library has been translated to form a third library which is compatible with this compiler.

8.4.5 WBC Equipment Installation

The satellite antenna for the Wideband Network has been installed on the roof of the building which houses ISI. Many legal and procedural difficulties had to be overcome to accomplish the installation. The rest of the station equipment, including the Earth Station Interface (ESI) and Pluribus Satellite IMP (PSAT), has also been installed and is being integrated.

The initial host system to be connected to the PSAT is a PDP11/44 running the Mini-Concentrator software. The PDP11/44 is a recently announced product, with delivery of ISI's machine scheduled in FY81. In the meantime, a PDP11/45 is available as a substitute.

The WBC project has also procured a UMC-Z80 programmable peripheral device for the PDP11 which, when combined with a Distant Host 1822 Interface supplied by Lincoln, acts as the interface to the PSAT. This equipment has been installed in the PDP11, connected by a 150-ft cable to the PSAT, and tested.

Last, the WBC project has received the first of four radio-receiver/clocks being purchased for use in absolute time synchronization in the ARPANET, Wideband Network, and ARPA Internetwork in general. These clocks receive National Bureau of Standards radio station WWVB to maintain absolute time accurate within one millisecond and make it possible to measure absolute transit times across networks to evaluate their performance for real-time data such as speech and video.

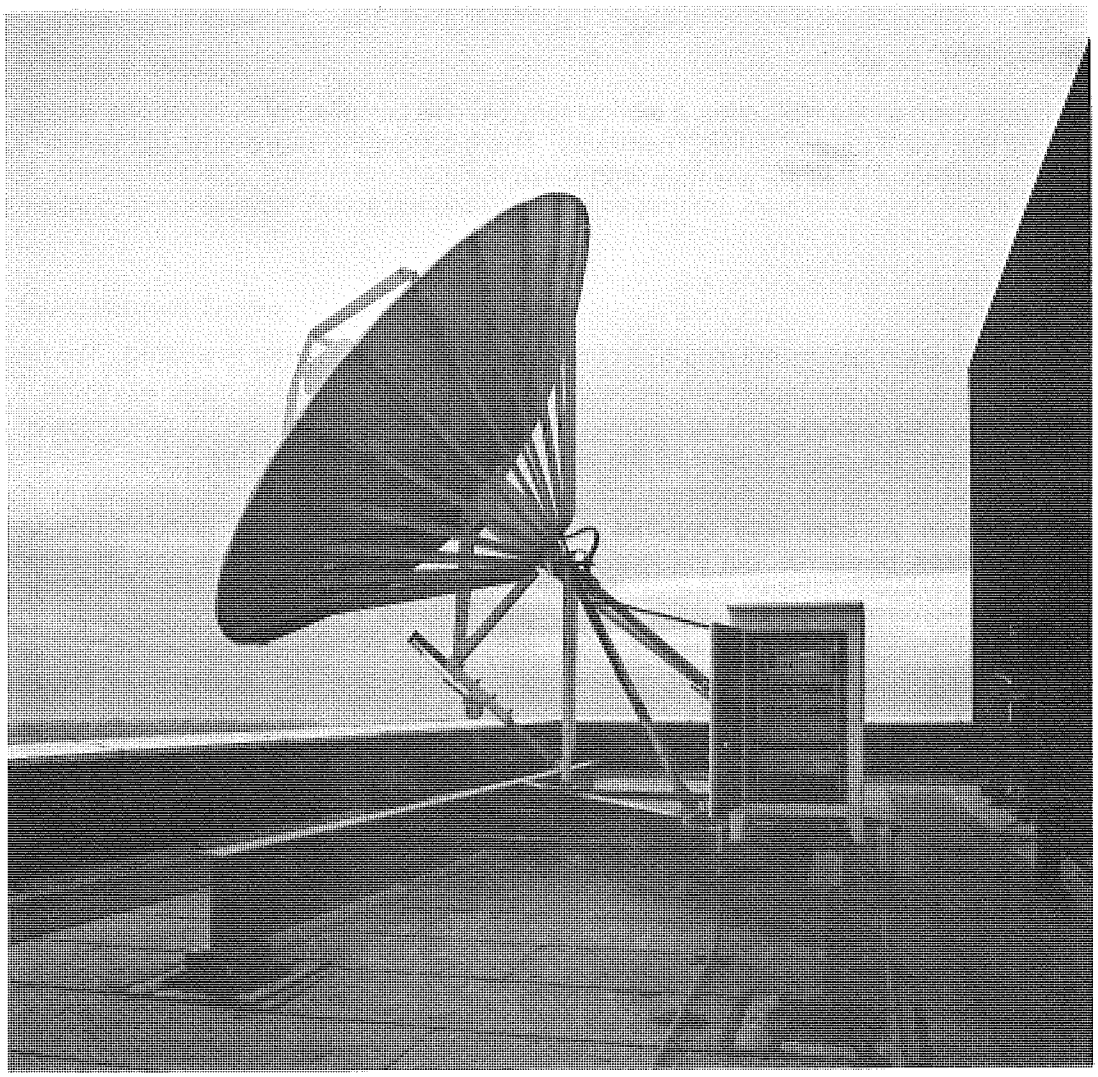


Figure 8-2: Roof-mounted earth station at ISI

8.4.6 Video Bandwidth Compression

As this year constitutes the transition from NSC to WBC, the video bandwidth compression effort has just begun. An experiment plan has been written for the investigation of video compression algorithms, including both technical and management plans. The investigation will begin with simulations of video compression algorithms, followed by evaluations of those algorithms for applicability in a real-time implementation, then the selection of compression hardware and the implementation of firmware for the chosen algorithm.

Examination of the options for the initial increment of video hardware has already begun, and a high-resolution video camera has been selected. The video bandwidth compression literature is being studied to find algorithms appropriate for use in the transmission of real-time video across the Wideband Network. Programs have been written for two cosine transform algorithms to compare their processing time requirements.

8.4.7 NSC Bibliography

Based on inputs from all the NSC sites, a bibliography [4] was compiled of publications outside the NSC community that were funded by NSC contracts. The bibliography was distributed as NSC Note 141 and also includes a list of all the previous NSC Notes.

8.5 IMPACT

As the ARPA NSC effort begins drawing to a close, its impact can be better evaluated. The project has produced good-quality low-bandwidth voice compression algorithms and made advances in the development of low-cost packet voice terminals. The work on speech bandwidth compression has significantly influenced the development of the single-chip LPC synthesizer by Texas Instruments. This in turn has led to intense competition from several manufacturers for the speech synthesis market. A complete vocoder also requires components to perform the analysis-half of the compression algorithm, of course, but the ready availability of synthesis devices has increased the demand for speech input (analysis) devices as well. Together, these developments will not only benefit the consumer market, but also bring down the cost of narrowband vocoders for secure military communications systems.

The ARPANET speech work also helped to define the performance characteristics which speech requires of a packet network. Existing network protocols concentrated on reliable delivery of each packet of data. For voice, however, it is more important to minimize the end-to-end delay through the network even if a few packets are lost or damaged. Two changes were made for packet speech:

1. The Network Voice Protocol was developed (primarily by ISI) to be used in place of the standard Host-to-Host protocol.
2. The ARPANET itself was modified to provide "uncontrolled packet" service which does not force end-to-end retransmission of packets for reliability.

With these developments, packet speech became a reality on the ARPANET.

The Atlantic Packet Satellite Experiment (APSE) contributed experience with PODA protocols for control of the broadcast satellite channel. Pre-allocated bandwidth "streams" were used to allow speech transmission with only one satellite-hop delay, and distributed conference control algorithms were tested to reduce floor handoff delays. Like the ARPANET, though, the bandwidth in the APSE was limited.

The Wideband Satellite program itself is a result of the impact of the ARPANET and APSE speech programs. Since it is a cooperative program between ARPA and DCA (Defense Communications Agency), the Wideband effort provides a real opportunity to help shape military plans for future secure communications systems. The increase in bandwidth which the satellite provides will allow experimentation not only with several voice channels but also with other media, such as video, graphics, text and facsimile. Eventually it will be possible to communicate between command centers in a teleconferencing mode for improved control of crisis situations. Even in less demanding circumstances, such as in the conduct of everyday military business, multimedia conferencing could reduce the need for travel and increase the productivity of the reduced military manpower available today.

8.6 FUTURE WORK

The Wideband Communication Program is the first development of high-bandwidth, long-haul packet communication. It provides significant opportunities for research in satellite-based, real-time, multimedia communication. Since the project is just beginning, most of its work lies in the future.

8.6.1 Initial Network Testing

Much of the effort expended during FY81 at all the Wideband Network sites will be to verify the operation of the network itself, and then to test and refine the transmission of multiple channels of packet speech across the network. The Experiment Plan prepared by Lincoln Laboratory includes a number of experiments which will continue throughout FY81 and beyond. ISI will participate in these experiments, including the design of tests, the collection of measurements, and the analysis of data.

As packet speech transmission becomes more stable, conference control mechanisms will be tested. The long delay of the satellite link causes problems for synchronization of control in interactive conversations. The ISI WBC project will participate in the implementation and testing of the various mechanisms.

8.6.2 Switched Telephone Network Interface

The test firmware which has been written for the STN Interface's microprocessor will be extended to the full operational capability. Ten "production" copies of the interface board will be built and distributed to some of the participants in the Wideband Program.

ISI expects to receive from Lincoln Laboratory a copy of its local access network (LEXNET) and two to five Lincoln Voice Terminals to be attached to the LEXNET. The WBC project will implement the necessary software to drive the STN interface as part of the Lincoln Voice Terminal and will also assist other sites (such as SRI) in the interfacing of the STN board to other equipment.

Once the transmission of packet speech on the Wideband Network becomes stable, and use of the STN interfaces for everyday calls becomes common, it may be desirable to increase the number of available channels and to free the Lincoln Voice Terminals for other experimentation. In that case, it might be desirable to implement voice terminal functions in the BBN Voice Funnel itself, and interface STN boards directly to it.

8.6.3 Video Bandwidth Compression

A major portion of the ISI WBC project effort will be the development of a video bandwidth compression system for use on the Wideband Network. The goal will be to build a system capable of transmitting color video with moderate motion in real time at a data rate of 1.5 megabits per second. Unlike voice, video is a new area of experimentation for ISI. Consequently, some initial work will be required to set up equipment and a software framework before experiments with the Wideband Network can begin.

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 into a "frame memory" to which special-purpose bandwidth compression hardware can be attached. A similar frame memory at a destination site will allow the use of a slow frame transmission rate by repeatedly displaying the same frame on the destination monitor. Figure 8-3 shows the components of the video transmission system. Included in the diagram are the transform processor and the interframe processor which will be null in the initial configurations.

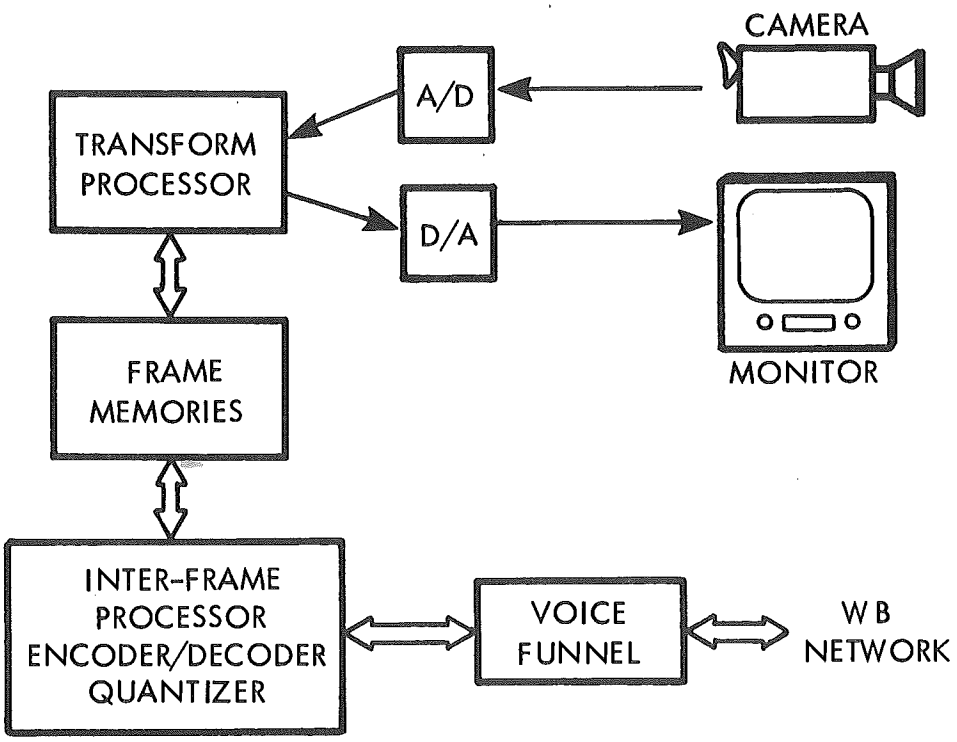


Figure 8-3: Narrowband video system

The video algorithm development will have the following major facets:

1. Early demonstration of a "frame-grabbing" transmission capability over the Wideband Network. Little or no bandwidth compression will be used. The rate at which frames will be sent will be strictly a function of the number of bits per picture element (pixel), the image resolution used, and the available channel bandwidth. This phase will provide a means for early testing of protocols, equipment, and channel performance.
2. Using the same set of equipment, several algorithms will be evaluated for the purpose of choosing a best algorithm or small family of algorithms to be implemented in real time. Evaluation will be done on the basis of image quality, suitability for economical real-time implementation, and how well the algorithm(s) matches the performance of the wideband channel, with performance in the presence of channel errors to be an important consideration.
3. As a result of the experience gained in the first two steps, an algorithm or family of algorithms will be implemented in real time, if such implementation is possible with reasonable effort and cost. Since such a real-time implementation will require building or buying high-speed hardware, it is hoped that the general characteristics of the

algorithm(s) will be defined well enough early in the course of the above evaluation step so that the hardware may be specified and construction or acquisition started early on. Thus the hardware would be on hand when the final algorithm is chosen.

8.6.4 Multimedia Conferencing

After gaining some experience with speech and video on the Wideband Network, and after the network conferencing protocols have been developed further, the WBC project will develop a multimedia conferencing system to include:

- Speech
- Video
- Graphics
- Text
- Facsimile

Before graphics and text can be integrated into a conferencing system, experiments will be required to determine the effect of the Wideband Network on their transmission.

The WBC project will first build a prototype voice/graphics conferencing system based on the SPEECH conferencing system and the graphics system of the ISI C2 Graphics Project. This system will provide experience with the control protocols and data mechanisms required to later build an integrated multimedia conferencing system.

After the hardware and software for transmission of real-time packet video have been developed, the video medium will be integrated with the voice and graphics into the conferencing system. Separate processors will be involved because video bandwidths cannot be handled by the speech or graphics systems. However, the video data stream will probably be controlled by the PDP11 which also supports the speech system.

8.6.5 Bulk Data Transfer

To investigate the transmission of nonreal-time data on the Wideband Network, the WBC project will develop software to offer the satellite channel as an alternative to long paths through the terrestrial ARPA network for bulk data transfers. This facility could be provided in steps:

1. A separate FTP program which transfers files over the satellite channel could be offered to users as an alternative. It might still be most reasonable to maintain the TELNET part of the connection over the ARPANET, and only use the satellite network for the data connection.
2. After the separate FTP program has been tested, the standard FTP program could be modified to transparently make a choice of ARPANET or Wideband Network for data transfer depending on the size of the file to be transferred and the loading of the two networks.

The protocol to be used for the FTP will probably be (at least initially) the standard FTP/TCP/IP combination. Experiments will be conducted to determine how throughput is affected by the size of the flow-control window and the use of additional protocol mechanisms such as negative acknowledgments for packets containing corrupted data but intact headers.

REFERENCES

1. Casner, S. L., EPOS system overview. Forthcoming as a USC/Information Sciences Institute technical report.

2. Gitman, I., and H. Frank, "Economic analysis of integrated voice and data networks: A case study," *Proceedings of the IEEE* 66, (11), November 1978, 1549-1570.

3. Markel, J. D., B. T. Oshika, and A. H. Gray Jr., "Long term feature averaging in speaker recognition," *IEEE Trans. Acoustics, Speech and Signal Processing* ASSP-25, (4), August 1977, 330-337.

4. Mellstrom, G., and E. R. Cole, *NSC Bibliography and Index to NSC Notes*, USC/Information Sciences Institute, NSC Note 141, October 1979.

9. DISTRIBUTED SENSOR NETWORKS

Research Staff:
Danny Cohen
Jeffrey Barnett
Iris Kameny
Yechiam Yemini

Research Assistants:
Richard Brooks
Avishay Halavy
Daniel Schwabe

Support Staff:
Debe Hays
Rolanda Shelby

9.1 INTRODUCTION

Military and commercial organizations are looking forward to large integrated data processing systems in the 1980s. Such systems will 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," i.e., 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 has focused upon Distributed Sensor Networks (DSN). A DSN contains a geographically distributed set of sensors (not necessarily all 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 and provide 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 will 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 [18].

9.2 PROBLEMS BEING SOLVED

Even though we believe that 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 were directed toward the more general problems: in particular, system issues that were both sensor- and scenario-independent. Sensor independence was possible because diverse sensor families behave similarly when their behavior is abstracted [9]. Scenario independence was dictated by our desire to examine large design and problem spaces encompassing many applications.

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

9.2.1 The Communication Problem

A DSN is an integrated system in that the total system, including the communications mechanism, works toward a common objective. This is unlike a typical system, in which the communications mechanism provides services to a variety of unrelated applications competing selfishly for resources. Further, a DSN is a realtime system where the application processes 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 goal of DSN communication is to provide for the interaction of application knowledge and communication for the good of the entire system.

9.2.2 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.

9.2.3 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 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: 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.

9.2.4 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 and correlated. It is also possible that 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 system-generated information and control the system toward fulfillment of an overall goal.

9.2.5 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 used to compute 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. Distance measurement can also be used to determine the present location of craft, particularly those engaged in surveying missions and thus needing an accurate measurement base.

The abstract position-location problem is the following: 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.

9.2.6 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 follows: Given a class of system organizations and restrictions on the communication facility (e.g., number of messages), determine when it is 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.

9.3 GOALS AND APPROACH

The major goal of the DSN project was to develop and export technology aiding the implementors of distributed and decentralized systems. The first step in this direction was derivation of an adequate description of the DSN problem space. This included 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, we wished to develop objective methods of evaluating a system's performance.

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

1. Simulation--describe and make operational approximations of system components.
2. Testbed--provide an environment for testing some components while simulating the behavior of other components.
3. Analysis--mathematical description and solution.

An aircraft can be simulated by writing sets of partial differential equations (obviously approximations) describing the behavior of the components and 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 developed 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 proposed 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) were always simulated. System components that are software processes could either be simulated or the actual program run. The testbed was our major design analysis tool. Provisions were 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. These include 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), including online debugging and probing by the simulated system user.

The testbed was basically an event-driven system with special provisions for handling the problems of local resource contention occurring at the system nodes. Communications modules handle global contention that occurs with the use of the available communication bandwidth. Since the testbed is not committed to a single communication paradigm (e.g., TCP or HEARSAY II), the testbed user must supply these modules.

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

9.4 PROGRESS

The activities and accomplishments of the DSN project are described below. Since this is the final report for the DSN project, a summary of progress for its entire duration is given. A collection of working papers written by project members is available in [10]. Some of the papers also appear in the open literature or have been presented at conferences.

9.4.1 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 [18] and [13], 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 [20] 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 [9], many different sensor families are reviewed, the conclusion being 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 this description allows us to investigate the high-level system issues without committing large amounts of resources to detailed sensor simulation.

9.4.2 Progress in System Methodology

The architecture problem for distributed systems has been a core computer science topic for many years. See, for example, [1, 14, 15]. However, the more complex issues of decentralized control and decision-making have received less attention. Several simple simulation exercises have been attempted with prior versions of our testbed. 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 that 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 assigning roles to players in a way that best satisfies the system's objective. The objective describes aspects of necessary system behavior and tradeoffs among competing alternatives.

We have made a literature search in the area of plausible inference to determine those techniques available for use in DSN--see [7]. Further, an efficient methodology has been developed for some classes of statistical reasoning in a distributed environment [8]. The work is based upon the developments of Glenn Shafer and caters to environments where multiple observations are available and confidence levels vary over time.

9.4.3 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 partially implemented. 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 is implemented in SIMULA. 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.

A configuration language is part of the testbed. It is used to describe (1) system topology including the initial configuration, (2) constraints on changes over time, and (3) the types and quantities of information to be made available to the experimenter. This form of system description is based on the capability model developed in [4].

9.4.4 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 [17] 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 conclusion results from 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 other processes that can supply needed input or employ their output to advantage. In [16] a naming scheme is proposed based upon application-determined labeling. 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 [11], 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 criteria to be expressed? This theme is continued in [21], where the problem of formal mechanisms to reason about protocols, particularly their time-dependent behavior, is discussed.

An important theoretical result is derived in [22]. 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 [12], the problem of flow control in distributed realtime systems is investigated. The conclusion is that there are some problems that cannot be solved without application-level knowledge. In particular, there are (at least) two different kinds of messages: those whose value decreases 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 [23]. 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.

9.4.5 Progress in Position Location

Our point of departure is that the task of position location would be implemented in packet radio networks (PRN) which would not wish to commit to it more than a minimal amount of the PRN's 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 the relative positions of nodes in a group can be computed; how the positions of positionable groups can be computed. In addition, we have obtained a rich set of theoretical results of major significance to problems of position location [19, 24, 25, 26]. 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 solution to any theoretically solvable position-location problem.

9.4.6 Progress in Distributed Computation Theory

A class of evaluation (estimation) algorithms called E-functions is 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.

Other 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.

9.5 IMPACT

The DSN project has had a commitment to studying the problems of distributed and decentralized systems. As some of our goals have been met, technology has been made ready for transfer from the realm of research to practical engineering development. Some examples of technology transfer are guides to evaluation of alternative systems, a capability model to describe distributed systems, results

on limitations and optimal utilization of communications resources, and position-location algorithms as part of packet radio networks. Other results remain theoretical or incomplete.

Besides the above, much of importance has been learned in general about DSN. Perhaps the most important is the definition and characterization of this problem area. We have learned that it is so vast and amorphous that many parameters need to be bound before pithy and specific guides can be provided to the implementors of actual DSN.

REFERENCES

1. Barnett, J., "Module linkage and communication in large systems," in R. Reddy (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, April 1979.

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

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

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

7. Barnett, J., Plausible inference references. Unpublished list of current literature on plausible inference.

8. Barnett, J., "Computational methods for a mathematical theory of evidence," in *Proceedings of the International Joint Conferences on Artificial Intelligence*, 1981. Forthcoming.

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

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

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

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

13. Davis, R., ed., "Report on the workshop on distributed AI," *Sigart Newsletter* (73), October 1980, 42-52. See the section by J. Barnett entitled "The USC/ISI DSN Project."

14. 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.

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

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

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

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

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

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

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

22. 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.

23. 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, April 1979.

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

25. 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, April 1979.

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

10. PERSONAL COMMUNICATOR

Research Staff:
Tom Ellis
Steve Saunders

Support Staff
Joyce Reynolds

10.1 PROBLEM BEING SOLVED

There is a definite and growing need for location-independent, secure, frequent, personal access to communication and information processing. Portable, versatile, simple-to-use communication devices are central to any plan for meeting this need.

Already, much essential data is accessible only through computer communications. The increasing use of computers at all stages in the acquisition, transmission, processing, arrangement, and presentation of information increases the volume of time-critical information potentially available to users. The need will become acute for many individuals in the military services to be provided with location-independent access to this data in performing their duties.

In order to provide such access, a communication device is required that can

- remain with the person at all times, without being in his way when he is not actually using it;
- be operated simply, without requiring much specialized training;
- handle enough different kinds and modes of information that he can rely on it for substantially all of his communication needs.

A device satisfying these criteria can well be called a "Personal Communicator" (PC).

The PC is just one component of a communications and computation network; all components must be present for the full utility of the concept to be realized. Communications connectivity, in particular, must be supported by other kinds of components. The ARPA Packet Radio Network (PRNet) is the prototype of such support for highly mobile, reconfigurable, survivable communications; it has provided the context for the exploration of the personal communicator concept.

Specific designs for the components of the system--terminals, repeaters, concentrators, computers, and databases--will depend on the ways these components interact; the interactions within such a system can be expected to differ sharply from current systems. We can know how to design portable communicators to correctly take advantage of the opportunities only after experimenting with some trial realizations of the concept.

Constructing the personal communicator device is a step toward realization of the whole system. The main value in early construction of model devices will be the exploration of system issues that it will allow, and the experimental focus it will engender.

10.2 GOALS AND APPROACH

10.2.1 Goals -- Experimentation with a Model Personal Communicator

The fundamental goal of this effort is to gain the understanding that must guide and support future designs of operational portable communicators and their associated systems. This guidance will be in the form of explorations and experimental implementations of alternative approaches to communicator design and components.

We have been concerned so far in this project with

- conceptually exploring the issues and constraints surrounding the concept of a highly portable, highly capable personal communicator, and
- building a suitable base for experimentation consisting of a model PC and a supporting software and communications context in which to pursue the more fundamental experimental goals.

The conceptual exploration has already resulted in an issues document and in a summarized functional specification condensed from the issues studies, and in guidance via personal interaction to workers elsewhere building advanced display devices.

10.2.2 Approach -- Requirements for a Model Personal Communicator

From our previous work we can characterize here what has to be packaged to support experiments. The overall characteristics must be such as to form a valid model of the functional and operational behavior of reasonable personal communicators; this means full portability within the experimental environment, fully functioning display, voice, and input capabilities, and fully functioning software to implement the required local and remote system behavior.

Physical requirements. Although in commercial parlance today "portable" may mean only that the manufacturer chose to attach a handle, we mean something more specific: a portable communicator 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, conveniently out of the user's way while he works, yet at hand when he needs it. We believe that this level of portability is essential to the full exploitation of the potential of packet radio. For example, a wristwatch fits this concept without question, as do pocket calculators and side arms, but backpack- or briefcase-sized things do not, since they cause excessive physical distraction or inconvenience when not in use. We believe that this attribute of unobtrusive portability is an essential ingredient to the notion of a truly "personal device."

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 there is a clear need for fully portable units; it is rapidly becoming feasible to build experimental models, and near future technology will support production designs. The pieces not yet available are low-power flat displays and miniature packet radios.

If the PC is to be extremely portable, it must not be too large, too oddly shaped for convenient carrying, or too heavy. We have built a mockup for a PC 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 or holster. The volume is sufficient to contain a display 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. More experience with portable operation will be required to establish where the optimum form lies within the reasonable range.

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 PC-size package is imposed by the available battery energy density. Low power circuits and design techniques, such as low duty cycle operation of subsystems, will be powerfully favored in all design tradeoffs.

Terminal parameters to satisfy functional needs and support experiments.

1. Display: The PC must be able to show as much text (24x80 characters, with lower case) as present CRT terminals and minimal pictures--480X240 pixels at the very least. Page replacement time for the display must be less than three seconds on psychological grounds and for many applications should be under one-tenth second. Either selective erase capability or fast update is required to allow the use of dynamic, informative cursors for feedback on user selections.
2. Voice I/O: Specialized hardware is required to digitize voice, amounting to perhaps 5 IC chips. Memory will be occupied to buffer speech segments of sufficient length to express a thought of meaning or to use a variable-delay communication net in real time.
3. Graphic I/O: There must be provision for a user to annotate or compose pictures and edit them together with voice and text. A touch panel or similar device is needed for pointing, drawing, and selecting displayed items. The processor must be powerful enough to interpret and display highly compressed graphic data received from the net within delay limits, and to compress graphics for transmission.
4. Memory: The terminal will be required to store several kinds of information for varying lengths of time--buffered speech and graphic data, messages for playback, display sections for quick scrolling or paging, and application programs. The 64K byte addressing range of present microprocessors will allow enough memory for experiments to establish memory requirements of operational units.
5. Communications interface: The speed of this interface must accommodate the data rate of the network for short bursts at least, and be capable of handling 16Kbit/second speech data on a continuous basis.
6. Processor: Present 8-bit CMOS processors are adequate for experimentation. We expect that more processing power will be required for operational PCs and will be available in the latter years of this effort. The main processor will be backed up by special processors as necessary for voice handling, graphics order execution, etc.
7. Secondary memory: Although it is easy to imagine uses for it, the requirements for long-term, large-capacity local storage are not well established by prior work. Again, experimentation is necessary to determine its cost-effectiveness in this kind of terminal.

10.3 SCIENTIFIC PROGRESS

The major accomplishments of this project are briefly listed here, followed by more discussion of the ones achieved during the period covered by this report. To date we have:

- explored in paper studies the issues of synchronization, communication data rates needed and prospectively available, pointing device technologies (touchpanels, etc.), and graphics capabilities and support;
- written a personal communicator functional specification intended to guide the developers of the radio unit and supporting systems;
- identified the pacing state-of-the-art elements and technologies (such as electrophoretic displays, low-power high-voltage drivers, and CMOS memory) which could maximally impact the problem areas;
- developed electrophoretic (EP) display-cell materials and structures, construction, and filling techniques, and built a number of test cells which have exhibited desirable characteristics in life, particle migration and switching parameters;
- investigated several concepts for addressing the pixels of an EP display, and identified the tradeoff issues among them. These approaches have included the triode internal structure, thin film transistor matrices and zinc oxide (ZnO) varistor switching. The tradeoff elements are mainly feasibility or yield at the current state of the art, addressing speeds, and contrast.
- developed a transparent touch-sensitive pointing technology that offers extremely low volume, weight, and power consumption;
- experimented with ZnO varistors as display matrix devices, including designing and building electrical and mechanical test equipment, mathematical models, and test circuit layouts to help evaluate the problems associated with this approach;
- built a benchtop breadboard version of the terminal system hardware, using a CMOS microprocessor with 64K bytes of memory, a CRT simulating the full-size display, and three variable-rate communications ports;
- evaluated several languages and approaches to programming 8-bit microcomputers, resulting in the selection of FORTH to support development of the software components of the PC.

10.3.1 Varistor Matrix Modeling and Experiments

Experiments were performed in order to obtain a first-hand understanding of the relative feasibility of using the threshold properties of ZnO varistor ceramic as a selection material for EP displays. Experimental materials were obtained from General Electric, where ZnO varistors are employed in surge suppressor products and are being investigated to drive liquid crystal displays. After characterizing the material parameters, modeling characteristic geometries, and testing specific cell designs, we determined, with currently available materials, that:

- excessive capacitance would not be a serious problem (contrary to the case with lower-density display arrays designed by others); but that
- large enough capacitances to allow charge storage for each pixel were not attainable without adding a thin-film dielectric to the process sequence, complicating fabrication considerably.

10.3.2 Design of a Model Personal Communicator

In order to meet the needs for experimental operation and development of the PC and to match what we believe from conceptual explorations are the proper performance parameters, we are building a model communicator with the following internal architecture.

The digital electronics consist of an RCA 1802 8-bit CMOS microprocessor, 64K bytes of RAM and ROM, a specialized processor for speech, a touchpanel circuit made from a monolithic A-D converter, and appropriate display drivers all connected via a common parallel bus. The processor executes 8-bit instructions at about 400,000 per second.

The performance of this model will be sufficient to explore the issues and establish the proper sizes and performance parameters needed in operational communicators. We expect that experience will lead us to increase the performance in some areas; some performance increases will be almost automatic as more powerful 16-bit CMOS microprocessors, already announced, become available.

The electrical power consumption of such a configuration, using published figures for available components, is no more than 324 milliwatts during active operation (while sending or receiving speech, computing graphic representation, etc.), 92 mW while idle between operations, and 23 mW in a standby mode (listening for net or user activity but not "in use"). These figures include processor, 32K of RAM, 32K of ROM, and the touchpanel; they do not include the speech processing or display drivers. Speech processing power requirements have not been established, but there is no reason to believe that they must be larger than 100mW active and 5mW standby. Display drivers are an item of serious concern, since so many circuits are required. Their power requirements will depend on the voltage levels, speed, and storage requirements imposed by the specific display technology and cannot be established with any precision at this time.

The breadboard model in its present state includes the 1802 processor, 64K of RAM, three programmable high-speed serial communications ports, and a bit-map CRT display subsystem, all using a standard iSBC/Multibus bus backplane for flexibility of future development.

10.3.3 Software Basis for Experimentation

After surveying possible programming arrangements, including several resident language systems such as BASIC and nonresident arrangements such as cross assemblers on the PDP-10s, we have selected the FORTH programming language and operating system to serve as a basis for our software development. FORTH is a resident system including a stack-oriented, modular programming language, secondary storage management, and numerous facilities for program development and maintenance.

We plan to use one of the high-speed lines to implement secondary storage for the 1802 on the PDP-10 disk file structures, thus centralizing our storage while retaining the benefits of a resident programming system on the model communicator hardware. This will involve us early in network-related software development, as we must replace the software floppy-disk driver program supplied as part of the operating system with a serial-line driver of our own plus a file-transfer protocol adapted to the secondary storage traffic generated by FORTH.

10.4 MILITARY IMPACT

We believe that the style of communications we propose to investigate will have a great impact on the effectiveness of many military missions. To verify this thesis, in previous work this project constructed a physical model of a portable terminal as a means of focusing conversations about the concept on the issues affecting future designs. Interactions with military personnel and personnel associated with military programs have met with high enthusiasm at the prospect and have prompted very useful suggestions on scenarios, functional capabilities, and interface styles. Application areas discussed have ranged from equipment maintenance support to a tactical field personnel command and reporting capability. Functional and interface capabilities, the inspiration of much of the application interest, result from the inclusion in one device of pictorial and voice interaction as well as text. This allows a choice or mixture of media to suit the occasion, i.e.,

- Text (either narrative or tables) is lasting, unambiguous, and not easily subject to misinterpretation due to various kinds of competing noise.
- Pictorial data is advantageous for quick interpretations of complex situations and lower level user search and selection. Tactical scenarios have been enhanced by the projected ability to retrieve maps from remote databases with up-to-date situation overlays and in some cases asking the user to contribute to the currency of the overlay database by graphical or textual additions to it from field locations. For equipment maintenance in the field, exploded view drawings annotated with up-to-date maintenance bulletins obtainable from a central database could significantly improve the effectiveness of this function.
- Voice is most natural for quick, easy, and interactive communication. Voice is also very useful in accompaniment with other medium such as pictures and tables for explanation or discussion.

Specific areas expected to benefit are:

- Command mobility--through more flexible communications;
- Man-machine interface design;
- Force effectiveness--through better command communications;
- Support functions--through better reporting and documentation availability.

10.5 FUTURE WORK

Real guidance for future designs will come from answers and partial answers to some research questions that we have identified and clarified through our previous work. By this time we have fairly strong ideas about how some of these issues should be handled but feel that experimental verification is appropriate.

10.5.1 Research questions

- How does one terminal find the address or identifier of another? How is a call initiated? How is the recipient notified? How are busy-signals, priorities, interrupts, and retrying handled?
- How should multiple media be combined? What structures allow for efficient, fail-safe synchronization without user concern and for editing before transmission?
- Where in the network--at the terminal, in a host, in a network relay node--should the processing and storage to implement each of various functions reside? What is the correct system architecture?

- How should a display of limited size be shared among multiple, competing, real-time user tasks? How can the user control this sharing?
- What are the proper uses of two-way conversational connections versus one-way messages? How can the two be smoothly integrated so that the user can shift from one to the other easily (e.g., if connection is broken, or a call is interrupted by higher-priority traffic)?
- How should access to databases be integrated with person-to-person messages and conversations?
- What methods are appropriate for authenticating users when calling, sending messages, and receiving messages? How can cryptographic key management be kept from interfering with the user's task?
- What aids can be provided for diagnosis of problems with the terminal or network, for temporary circumvention of faults, and for repairs? How can these be constructively presented to a user who is not familiar with the internal construction of either the terminal or the network?
- How should various pieces of status information--time, network connectivity, host availability, user position, etc.--be presented so as not to be missed if important, but not to be unduly disturbing to ongoing tasks?
- How should the PC connect with the radio network and other communication modes (if any)? I.e., what is the proper bandwidth per user and how can it be attained?
- What user-input devices are appropriate for a handheld communicator (keyboard, touchpanel, joystick, tiltmeter, accelerometers, etc.) and how should they be configured?
- What is the optimum physical size and shape for a PC, given the level of technologies it is to be built from? Should less-often-needed capabilities be provided via modular plug-together units, or should there be a series of integrated models each offering a different set of optional features?
- What tradeoffs can be made to achieve lower cost communicators without losing significant functionality? When does centralizing a function result in lower system costs?

Some of these questions have been investigated before in other contexts; we should not cover the full generality of such questions as user-input device comparisons, but rather concentrate on the specific application to a handheld communications device with local computation ability.

10.5.2 Steps to be taken

The answers to these and other questions will come from experimental investigations that implement and refine proposed solutions and evaluate the results. To perform these experiments we will need to build testbeds of hardware, software, and environment. The following steps will be required:

- build, operate, and refine fully-portable, full-function personal communicators;
- integrate the communicators with the communications networks at ISI, beginning with the ARPANET, to provide sources of messages, realistic operating conditions (including a rudimentary real-time message environment), and incentive for others here to use and criticize our models;
- design, build, test, and refine user interfaces and control procedures for the personal communicators, validating them through experiments with users from within and outside the research community.

This project has been kept at a relatively small level until the contributing technologies are sufficiently well along to support realization of an actual research testbed. Our emphasis has been in identification both of operational or functional issues, and of specific needs for progress in the

contributing technologies, in pressing for development in these areas. The major technological areas still falling short of the needs of portable personal communicators are in radio communications and in display characteristics. We have given the low-power, flat-panel display developers much attention and guidance in the past and intend to continue to do so as we feel that we can have and have had considerable impact here.

11. APPLICATION DOWNLOADING

Research Staff:
Robert Balzer
Alvin Cooperband
Martin Feather
Philip London
David Wile

Support Staff:
Joan Elliott

11.1 INTRODUCTION

The rapid advancement in the processing capabilities of computer terminals provides an opportunity to allow terminal processors to share in the execution of application programs. The downloading of interactive applications into a terminal may often result in a decrease in demand for shared, central processor time. However, developing such distributed application programs is a difficult task. This difficulty is mitigated by first allowing an interactive application to be designed, implemented, and tested using standard development techniques in a single-processor environment, then semiautomatically partitioning and optimizing the application in a distributed processor system consisting of a terminal and a host computer.

11.2 BACKGROUND

The state of the art of computer terminal hardware is advancing quite rapidly, and these capabilities are expected to continue developing at least as quickly. Naturally, users and implementers would like to utilize these processing capabilities, especially for applications which currently overload central timeshared computing facilities.

This overloading typically arises from neither excessive computing nor memory requirements, but rather from the need for responsiveness (providing feedback rapidly enough to present disruption of the user's train of thought). Building responsive large time-shared facilities has proved most elusive. In network-based systems, communication delays, by themselves, often preclude responsiveness.

In reaction to this difficulty, the notion of achieving responsiveness through dedicated local processing has received considerable attention (e.g., NSW front end processor). Hardware advances in computer terminals have made this option of dedicated local processing economically cost effective. In addition to relieving the responsiveness problem, local processors provide benefits of improved reliability by reducing dependence on centralized facilities and security by keeping sensitive data physically segregated from shared resources.

However, the largest potential benefits may arise from improvements in the man-machine interface. The use of local dedicated processors only addresses this larger problem (which is separately the focus of a major DARPA-IPTO effort) through increased responsiveness--but this may well be a central issue. For example, a new generation of two-dimensional CRT screen-based editors are being produced which are both more powerful and more natural than previous editors. Universally, these

editors require both more computational power and increased responsiveness for their effective use, and for this reason, they are being implemented on local dedicated processors. There is strong implication that improvements in the man-machine interface will require further increases in computing resources and responsiveness, which will necessitate dedicated local processors.

11.3 PROBLEM BEING SOLVED

The availability of local dedicated processors powerful enough to satisfy these demands cost effectively is not in doubt. The difficulty, as usual, lies in the lack of software technology to effectively utilize this processing potential.

The source of this difficulty is that the dedicated local processor cannot run the entire application, but only a portion thereof, with the rest being handled by a shared central processor. There are two reasons for this need to split the application between a dedicated local processor and a shared centralized processor. The first is functional. Some portions of the application deal with multiply accessible shared data and/or resource allocation of shared resources. This portion cannot be distributed among dedicated local processors, but must be maintained as a shared centralized capability. The second reason is resource limitations (mainly memory) of the dedicated processor. The entire application typically exceeds these resource limitations, and so the most appropriate subportions of the application must be identified for inclusion within the dedicated processor.

Building "distributed" systems which operate on cooperating processors is much more difficult and less well understood than building systems which operate in a uniprocessor environment. This difficulty has several sources. First, additional design complexities are introduced by the need to divide the functionality between processors and by the need to introduce communication and synchronization protocols between the processors to maintain compatible versions of data structures which are logically shared. Second, additional implementation complexities are introduced by using two different processors, with (typically) two different languages and programmer support systems. Third, debugging is greatly compounded by use of more than a single processor (as weak as debugging technology is for single-language single-machine programs, it is far weaker for multilanguage multimachine situations). Finally, application maintenance is clearly compromised by the additional complexity introduced.

11.4 APPROACH

These difficulties, introduced by the necessity of building applications distributed between a local dedicated processor and a shared, centralized processor, could be largely, or completely, mitigated if we successfully developed a technology to split an application apart AFTER it had been designed, implemented, tested and debugged in a uniprocessor environment. Splitting the application and assigning the parts to the separate processors would simply be an additional optimization step to be performed after the rest of the software development had been completed. The tremendous benefits of designing, implementing, testing and debugging the entire application within a single language uniprocessor environment automatically result from splitting the application after these other activities are completed.

Since the testing and debugging would have already been completed, this technology would have to be guaranteed to maintain the validity of the implementation. Such a technology of implementing a

program via successive transformations, each of which maintains validity, was already being developed (at ISI and elsewhere).

Our approach meshed nicely with existing transformational implementation technology and can be viewed as a limited, specialized set of transformations which deal specifically with the issue of dividing an application among a host and terminal processors.

In our approach, the programmer would simply indicate which portions (program and data) of the application should reside in the local dedicated terminal processor and which should remain in the centralized shared host processor. Based on the specified split, the application would be transformed to substitute interprocess control protocols for those subroutine calls and returns which spanned processors, and interprocess data protocols for those data accesses which spanned processors. The programmer could also specify that some portions (program and/or data) should exist in both processors. This introduces the need for additional protocols to maintain data consistency between the processors [1].

Our contributions are described below in terms of the prototype partitioning system, relaxation of the partitioning requirements, and optimizing the introduced communication protocols.

11.5 RELATED RESEARCH

Considerable attention has been paid to partitioned applications in the area of computer graphics. In a typical graphics application, a small, dedicated satellite computer, which is used to drive a graphics terminal, communicates with a large, shared, host processor over a narrow-bandwidth line. The satellite computer is often used to perform some application-specific functions to reduce the load on the host computer or to reduce communications. Since the host and satellite computer typically use different languages and operating systems, most such partitioned applications have required the implementer to start with a partitioned design, use different languages on each side of the partition, and provide his own cross-partition interface to support passing of data and transfer of control. Hamlin [4] developed a system that attempted to avoid all of these difficulties. He reports some prior efforts in this area; in the ICT system [3] interactive satellite graphics functions and cross-partition communications are driven by interaction control tables supplied by the application programmer; the GRAPHICS-2 system [2] provides automatic cross-partition communication, but at the cost of duplicating the interactive graphics functions and processing; THEMIS [5] uses separate, specific languages for the host and satellite portions of the interactive graphics functions and requires the application programmer to specify the cross-partition communications; ICOPS [6] permits modules written in a special language to be assigned to the host, the satellite, or both with automatic transfer of control across the partition, but does not support references to variables on the other side of the partition. None of these earlier systems supported arbitrary applications. Hamlin's system (CAGES) permits an arbitrary application written in PL/1 to be partitioned so that a procedure can be assigned to either side, and it automatically implements cross-partition reference to global data, passing of procedure parameters, and transfer of control. Although Hamlin clearly identified the advantages of having a system that permits an implementer to develop an application in a single high-order language without committing himself to a particular partitioning of data and functions, his system is of only limited interest because it precludes passing any POINTER, BASED, or CONTROLLED variable types across the partition, thus severely restricting the programmer and limiting the way in which an application may be partitioned.

11.6 ACCOMPLISHMENTS

A prototype system has been developed to implement the methodology of application splitting described above. Partitioning of the application is based on user declarations and proceeds automatically. The resulting protocols for making interpartition control transfers require optimization to make them more efficient. These optimizations include reducing the number of interpartition transactions and reducing the amount of data that needs to be communicated from one processing node to the other. We are developing a library of such user-invokable transformations. A major issue, in addition to the discovery of the optimizing transformations themselves, is the support required by the user for effective use of such a transformation library. Such support includes analytic tools and instrumentation which aid the user in understanding his partitioned system, and a congenial environment in which he can optimize his program by selective application of the transformations.

11.7 THE PROTOTYPE SYSTEM

11.7.1 The Problem

In splitting an application from a single processor system to a multiple processor system, several different scenarios are conceivable.

- A system in which the functions from the original application were split into two groups. Each group runs on one processor. Data passing is "message passing," presumed to be infrequent and of small bandwidth. Two "subscenarios" are possible:
  - Each group of functions behaves as an autonomous unit; synchronization is through critical sections.
  - Function execution follows the control flow of the original application, in which there is always a single locus of control. Control will always reside in one processor or the other.
- A system in which one processor views the other as a data structure. Here, data flow dominates and control is trivial. This is sort of a "flat" version of the first scenario above.
- A system in which data is shared between the components, containing pointers back and forth between data spaces.
- A system in which data is shared between the components through a shared memory.

The protocol developed will depend considerably on the hardware available for a split, in general. However, we feel the most general protocol is one in which data structures are shared--perhaps maintaining copies on each machine--and in which control structures are potentially parallel. This characterizes the ultimate protocol which we wish to handle.

The present capability is a small subset of such a general control facility, yet it is considerably more general than several of the above scenarios. Both the set of functions and the set of data structures are partitioned to reside on a particular machine. No copies of data structures are maintained, but the data structures may reference each other across partitions. The original control structure is retained, in that the cross-partition control protocol allows recursive calls to the opposite partition.

11.7.1.1 System Configuration

The development of the prototype system has allowed the investigation of the issues involved in distributing data and control between two processors. We have ignored issues of memory resource limitations and I/O. The prototype system consists of two Interlisp [7] jobs running on separate ARPANET Hosts communicating via the Interlisp Net facility (currently simulated by independent Interlisp jobs on the same processor communicating via a shared file).

The prototype system, in addition to providing an environment for investigating optimizing transformations, was designed to limit the analysis required to produce a partitioned system from the user's partitioning declarations and with the idea of keeping the protocols simple--leaving the complexity to the optimizing transformations. The protocols were designed to permit LISP's critical EQ test (equivalence of pointers) to remain valid even when accessing remote data.

The system is organized about the activities depicted in Figure 11-1.

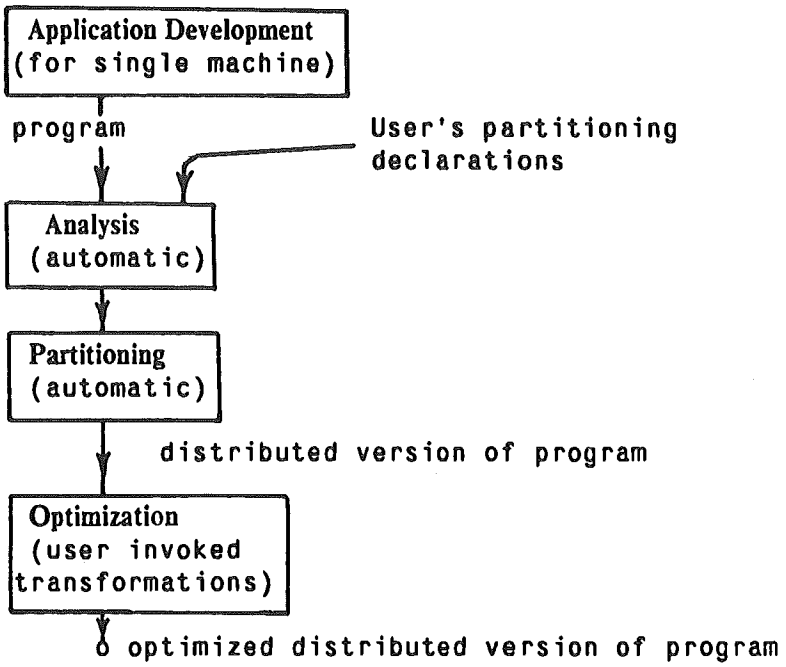


Figure 11-1: Development activities

The application programmer firsts builds his application. After stating partitioning declarations, the application program is analyzed. The partitioning declarations and the results of the analysis are fed into the partitioning system, which produces a distributed version of the original application program. Finally, user-invoked transformations are applied to the distributed application components for the purpose of optimizing the interprocessor protocols.

11.7.1.2 Application Language and Partition Definition

The application language for programs to be partitioned by the Application Downloading system is Interlisp augmented to allow user-defined types. All expressions are typed and the types of expressions are determinable by static analysis. Thus, the language would be strongly typed, except that union types are permitted.

The protocol for the partitioning basically divides all data structures and functions into three distinct classes: **private**, **shared**, and **owned**. Private objects are referenced by only one partition. Shared objects have no owner but have the same representation on either side of the partition. Owned objects are objects which reside on one side but are referenced on **both** sides. Owned objects have an owning partition: initially the ownership is static but we will later consider allowing ownership of objects to change dynamically.

In order to partition the data structures into the categories above--owned, private, and shared--the data **types** of the language have been divided into several categories. These in turn will be used to determine the ownership of the data **instances**:

1. **Primitive Types** include primitive LISP types such as numbers, atoms, booleans, strings.
2. **Enumerated Types** are types for which the instances are predeclared.
3. **Record Types** are data structures with predefined structure and a fixed number of named access paths. The field entries in Record Type objects are typed objects themselves.
4. **Union Types** are types whose instances are the collected instances of their declared subtypes.
5. **Structure Types** are data structures with predefined structure but without predefined access paths. Thus, access to substructures is with standard LISP functions. The elements of structures are themselves typed objects. Examples of structures include an array of numbers, a list of records.
6. **Unrestricted Types** include unrestricted LISP data structures implemented with CONS cells. Unlike structures, no clean structural element type can be discerned.

Currently, data is partitioned on the basis of types to eliminate the need for run-time determination of the accessibility of data (local vs. remote). There are restrictions on legal partitionings pertaining to the different classes of types. Types are declared to reside wholly in one partition or the other.

The treatment for each class of type with respect to legality of partitionings and interpartition referencing are:

- All primitive and enumerated types may be referenced remotely, without restriction.
- Unrestricted types may not be passed or referenced remotely. That is, instances of unrestricted types may be referenced only in the partition in which their type was declared to reside.
- Record types can be remotely referenced via a protocol described later. The protocol was designed to preserve LISP's EQ (pointer equivalence) test rather than just LISP's EQUAL (structure equivalence) test.
- Presently, structure types may not be referenced remotely. In section 11.7.2, we will discuss the difficulties involved in remotely transferring data not possessing fixed, named field accessors.
- All subtypes of a union type are required to be in the same partition as the union type.

This prevents the "almost strongly typed" nature of the language from being a problem. The system does not need to know the precise type of an expression as long as it knows the union type, since the value of the expression is then guaranteed to exist in the same partition as the union type.

11.7.1.3 Constructing the Partitions

The system builder designs his partitioning as described above, and then calls a function which constructs the partitioned version of the system. This partitioning function deletes the private functions and data declarations of the remote partition and redefines remotely owned functions and data accessors as calls on protocol functions to accomplish them. In particular, it is not necessary to modify the definitions of any locally owned functions or data structures, even if they are also accessed remotely (their definition in the remote partition will be redefined to allow such access).

To be more specific, assume we are constructing the partition for the **host**. All of the functions and data type declarations for the original system are present initially. Hence, we first remove all functions and type declarations which are private to the **terminal**. We then redefine each function which is owned by the **terminal** to be a call on a protocol transfer function: the name of the function and its parameters are passed to the remote-function-call function. Similarly, we redefine all accessors to record structures for records owned by the **terminal** as calls to protocol transfer functions which:

- Fetch data from a field of a remote datum;
- Store data into a field of a remote datum;
- Create a remote datum.

We then save the "core image" of the system with deleted and redefined definitions, after loading in a package which effects the protocol described below. This core image will communicate with its dual core image made by repeating the above process for the **terminal**. The nature of this communication will be discussed presently.

The functions mentioned above all behave in the following way: some **data** (function parameters or record instances to be accessed) are passed to the opposite partition and then some **task** is performed by that partition on the data. Usually, data are passed back as the result of such a task. Hence, the functions above may be characterized abstractly as involving a **data transfer protocol** and a **control transfer protocol**.

11.7.1.4 Data Transfer Protocol

The data transfer protocol implemented presently is quite simple. Since there is always only one owner of a datum, and since all remote access to it is through protocol functions, very little bookkeeping concerning a remotely accessible datum is necessary. Basically, the only information needed for a datum which is passed across the partition is a unique name for it and some means for guaranteeing that the name (when passed back) will refer to the original datum. Because of garbage collection in LISP, the address of a datum is not a suitable "name" for its remote reference: the datum may change positions after a garbage collection and the remote partition will not be aware of the change. We chose a simple scheme for maintaining names as indices to an array of remotely accessed data (see Figure 11-2). We call this array the **Entry Table**, and an index into it an **Exit Index**.

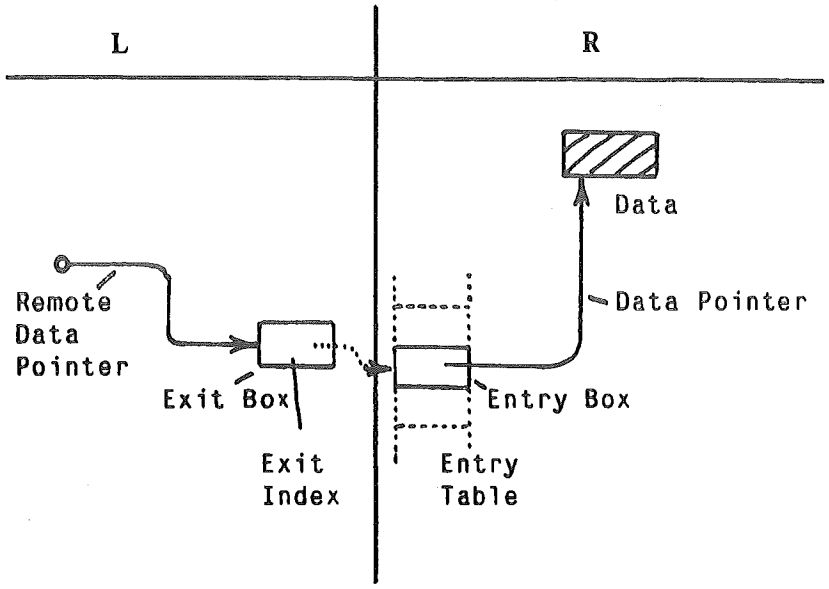


Figure 11-2: Remote data maintenance

Now, an implementation which merely treated the opposite partition's exit indices as integers would work. However, strong type enforcement would be required to guarantee that those integers were "safe." Hence, when an exit index arrives from the owner's side, the index is simply put into an Exit Box. The address of the exit box is then passed as the name for the remote datum on the non-owner's side.

In fact, this arrangement will be even more important for a **multimachine garbage-collection scheme**, where the implementation must have a handle on which integers represent external pointers. Although we have not yet implemented a garbage-collection scheme, garbage collection is easily accomplished for noncircular structures by intercepting the marking algorithm before an unreferenced Exit Box is deallocated and reporting it to the other partition. In fact, all Exit Boxes are linked together to permit searches which maintain EQ pointers on both sides: hence, the garbage collection mechanism must occasionally be invoked with these pointers "masked out" for the purposes of marking.

Circular structures (across partitions) are more difficult. We have developed an algorithm relying on "postulating" garbage, and then allowing the normal collection mechanism to proceed until it stabilizes. We have not tested this algorithm, nor do we intend to implement it.

Of course, this arrangement is symmetrical: there is an Entry Table and a set of Exit Boxes on each side of the partition. When a datum is to be sent across the partition, the protocol involved is based on its type:

- Locally Owned Datum: an exit index is passed as follows:

- New Local Datum: if this is the first time a reference to this datum is being made (because a new entry in the Entry Table was necessary), the message: **My New Data**[Entry Index] is sent.
- Old Local Datum: this item was already in the Entry Table. Its index is sent via the message **My Old Data**[Entry Index].

- Remotely Owned Datum: the value stored in the Exit Box is passed across in the message **Your Data**[Entry Index].
- Copied Datum: the datum itself is passed across verbatim, in the message **Our Data**[Datum].

Upon receipt of these messages, the opposite side must prepare the datum for its functions to work on it. This is done as a response to the message type sent:

- My New Data: a new Exit Box is allocated and the remote Exit Index is stored in it. The data is now represented as the address of the Exit Box.
- My Old Data: all Exit Boxes are searched to find the one which contains the Exit Index passed. The address of the one found now represents the datum.
- Your Data: the value stored in the Entry Table at the passed Exit Index location is the datum referenced.
- Our Data: the passed value is the value used.

11.7.1.5 Control Transfer Protocol

Our control transfer protocol allows the application to utilize the full recursive LISP control structure.[3] This recursive control regime allows interpartition calls to be nested within one another and results in the LISP control stack's being interleaved between the two partitions. To prevent free variable references from having to search this interleaved stack for dynamic bindings (by passing messages back and forth between the partitions), all potentially accessible free variables are passed across the partition when control is transferred so that all relevant portions of the stack are local and the normal LISP stack referencing mechanisms can be used.

The protocol itself is easiest thought of as a sequence of "information" messages followed by an "action" message. In particular, with each change of control which involves a function call or return, the set of free variables which are shared is updated. Then the call or return is made. Some calls cannot nest, and hence, no free variable maintenance is necessary. The current protocol does not need to update free variables for any remote record create, access, or store.

It is easiest to understand the protocol by following an example. Figure 11-3 is an abbreviated transcript of the Eight Queens problem using the running protocol. Arrows indicate the direction of information flow between the partitions. The **terminal** partition contains the top level function (Q8) and the function REMOTE INITIALIZE; the **host** contains the functions INITIALIZE GLOBALS,SAFEDIAGONALSAT, and PLACEQUEENSON. Control starts in Q8 which immediately calls a remote function. This is done by calling INITIALIZE GLOBALS which has been redefined in the **terminal** partition to a call on the remote-procedure-call function, passing its own name and its arguments to the function. The remote-procedure-call function simply transfers control to the other partition through the control protocol, which causes the free variables to be sent before the actual

[3] Actually, the "spaghetti stack" of Interlisp is not simulated: only the normal single control thread stack is supported.

```
                                                                    Calling
                                                                      Q8
                (FreeVariable ATTEMPTEDCOLS (OurData NOBIND))  |
                (FreeVariable EMPTYCOLS (OurData NOBIND))      | ==> HOST
                (FreeVariable NUMBEROFQUEENS (OurData NOBIND)) |
                (FunctionCall INITIALIZEGLOBALS NIL)           |    Calling
                                                                      INITIALIZEGLOBALS

            |   (FreeVariable ATTEMPTEDCOLS (OurData NOBIND))
TERMINAL <== |  (FreeVariable EMPTYCOLS (OurData NOBIND))
            |   (FreeVariable NUMBEROFQUEENS (OurData NOBIND))          Calling
            |   (FunctionCallREMOTEINTITALIZE NIL)                       REMOTEINITIALIZE

                (FreeVariable ATTEMPTEDCOLS (OurData NOBIND))  |
                (FreeVariable EMPTYCOLS (OurData NOBIND))      | ==> HOST
                (FreeVariable NUMBEROFQUEENS (OurData NOBIND)) |
                (ReturnRecord FunctionCall (MyNewData 1))      |    Returning from
                                                                      REMOTEINITIALIZE

            |   (FreeVariable ATTEMPTEDCOLS (OurData NOBIND))
TERMINAL <== |  (FreeVariable EMPTYCOLS (OurData NOBIND))
            |   (FreeVariable NUMBEROFQUEENS (OurData 8))               Returning from
            |   (ReturnRecord FunctionCall (MyNewData 1))                 INITIALIZEGLOBALS

                (FreeVariable ATTEMPTEDCOLS (MyNew Data 2))    |
                (FreeVariable EMPTYCOLS (MyData 1))            | ==> HOST
                (FreeVariable NUMBEROFQUEENS (OurData 8))      |    Calling
                (FunctionCall SAFEDIAGONALSAT ((OurData 1) (OurData 1)))    SAFEDIAGONALSAT

            |   (FreeVariable ATTEMPTEDCOLS (YourData 2))
TERMINAL <== |  (FreeVariable EMPTYCOLS (YourData 1))
            |   (FreeVariable NUMBEROFQUEENS (OurData 8))               Returning from
            |   (ReturnRecord FunctionCall (OurData T))                   SAFEDIAGONALSAT

                (FreeVariable ATTEMPTEDCOLS (MyNew Data 2))    |
                (FreeVariable EMPTYCOLS (MyData 1))            | ==> HOST
                (FreeVariable NUMBEROFQUEENS (OurData 8))          Calling
                (FunctionCall PLACEQUEENON ((OurData 8) (MyNewData 3)))     PLACEQUEENON

            |   (FreeVariable ATTEMPTEDCOLS (YourData 2))
TERMINAL <== |  (FreeVariable EMPTYCOLS (YourData 1))
            |   (FreeVariable NUMBEROFQUEENS (OurData 8))               Returning from
            |   (ReturnRecord FunctionCall (YourData 3))                  PLACEQUEENON

                (FreeVariable ATTEMPTEDCOLS (MyNew Data 4))    |
                (FreeVariable EMPTYCOLS (MyNewData 5))         | ==> HOST
                (FreeVariable NUMBEROFQUEENS (OurData 8))          Calling
                (FunctionCall SAFEDIAGONALSAT ((OurData 2) (OurData 2)))    SAFEDIAGONALSAT

TERMINAL <== |  (RemoteAccess (ROWNUMBER) (YourData3))
                                                                      Record Access
                (ReturnRecord RemoteAccess (OurData 1))        | ==>HOST
                                                                      Access Value
TERMINAL <== |  (RemoteAccess (COLNUMBER) (YourData 3))
                                                                      Record Access
                (ReturnRecord RemoteAccess (OurData 1))        | ==>HOST
                                                                      Access Value

          *     *     *
```

Figure 11-3: Control protocol transcript

control transfer occurs. Control then passes to the **host** partition with the message: (FunctionCall INITIALIZE GLOBALS NIL).

Then the **host** partition takes over. Each of the expressions passed is EVALed after invoking the data transfer protocol on its arguments. Hence, each free variable is set[4] and then the function INITIALIZE GLOBALS is called. This function itself does a call to the other partition, on REMOTE INITIALIZE. Of course, its definition has been changed to a call on the control protocol, so it goes through the same scenario as above.

Finally, REMOTE INITIALIZE returns. The free variables' values must again be copied (in case they changed) before control actually returns. Notice that as the process progresses, each of the free variables becomes initialized. In particular, the data transfer protocol messages described above are visible as arguments to the control messages.

11.7.2 Reducing Restrictions on Legal Partitionings

The restrictions imposed on legal partitionings described above are too constraining. Our experience has shown that forbidding structure types to be passed as data across the partition often makes the choice of a reasonable control partitioning difficult. In fact, it would be desirable to allow unrestricted access to data of all types. Let us consider the case of structure types, since unrestricted types are a generalization of structure types. The difficulties in dealing with structures arise from the choice of possible methods for allowing remote references. These choices are:

1. use a remote reference protocol as is done for record types, or
2. transmit across the partition a copy of the structure.

Both of these solutions have their individual problems.

Using a remote reference protocol leads to difficulties with updates and modifications to the data structure. In data structures without a fixed number of named fields (vis a vis record types), it is customary to use LISP primitive operations. Performing an update would require intercepting calls on LISP primitives and invoking a remote update protocol in their place. In addition, using a remote reference protocol would require an interpartition transaction for every modification desired. For instance, one reason for using a structure type which is a list of records would be to allow sorting. If the sorting operations occur in code which is remote to the location of the data, every modification would require a transaction--clearly unacceptable.

Copying the data structure to the remote partition requiring update access to it would solve the above problem. However, the copying could become an expensive data transfer transaction. Furthermore, the EQ (pointer equality) test is lost, and a determination of whether the data structure has been altered would have to be performed in order to determine if it need be sent back to its original owner.[5]

---

[4] Here, to the "uninitialized value" symbol, "NOBIND".

[5] Unrestricted types have the same problems as structure types. But they have the additional problem that a determination of their "extent" is difficult to perform.

One solution being considered for these problems with structure types is to build into the language a set of templates (e.g., LISTS, ARRAYS, TREES), each with built-in access, update, and iteration primitives. Thus, structures will be similar to records in that they will have named accessors, but the restrictions on fixed numbers of fields would not be imposed. This would provide a solution to the problem of intercepting calls on LISP primitives for the purpose of invoking the protocol interpreter. Structure references would then go through a remote reference procedure similar to that for records. If the inefficiencies involved in remotely referencing structures becomes overbearing, an optimization (described in the next section) can be applied which creates a local copy of remote data.

Another restriction--that all instances of record types or structure types must reside in a particular partition--is relaxed during the optimization phase (section 11.8.1) In the future we may allow the programmer to achieve this effect through declarations rather than optimizations.

11.8 OPTIMIZATIONS

In this phase of development, the distributed application components are optimized by programmer-invoked transformations. The purpose of this optimization is to decrease the run time expense of going through the protocol interpreter, creating/accessing/updating data across the partition, and transferring control across the partition. Three classes of optimizations are evident:

1. Data Optimizations, which adjust the residency of data to eliminate the expense of remote data references.
2. Code Optimizations, which adjust the control behavior to eliminate multiple transfers of control, multiple remote data references, or a mixture of both.
3. Run-time Activity Optimizations, which decrease run-time protocol activities if compile time analysis can demonstrate that they are unnecessary.

11.8.1 Data Optimizations

Residency of data will have a dramatic impact on the number of remote data references across the partition. An obvious improvement arises from making a local copy of a piece of remotely residing data if there will be multiple references to that object in the same partition; see Figure 11-4. This raises several issues; a mechanism is required to take a reference, determine whether or not there exists a local copy, and act accordingly. It must be possible to determine when the value of the local copy needs to be sent across to the remote side (e.g., updating the local copy, thus invalidating the remote version). Finally, the programmer must decide when the savings to be made exceed any extra expense involved in the local copy mechanisms (i.e., know when a local copy optimization would truly be an improvement).

A total relaxation of the notion of data residency leads to the concept of "floating" data, equally able to reside in either partition, and which "floats" across to the side making the most recent reference. We envision a mechanism for supporting both local copies and floating data.

Relaxing the partitioning restrictions on remote accesses to structure type data will induce the same set of optimizations on structures, with the extra degree of freedom of deciding how much of a whole structure should be locally copied (or floated).

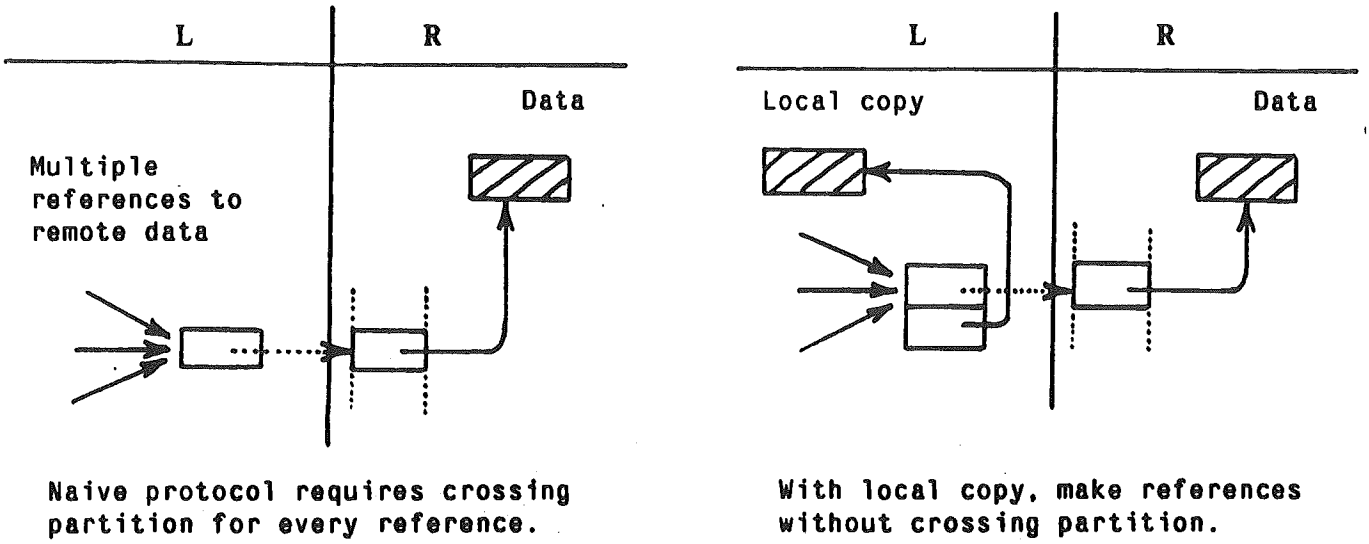


Figure 11-4: Local copies of remote data

11.8.2 Code Optimizations

This class of optimizations seeks to reduce the interpartition transaction expenses by adjusting the control flow rather than the data residency. We are developing a set of such manipulations.

For example, nested remote field accesses--suppose we ask for the x-field of (the y-field of Z), where Z is remote, and its y-field is also remote. The naive protocol would make two separate cross-partition accesses, which we could optimize to a single access making the nested access on the remote side. (See Figure 11-5.)

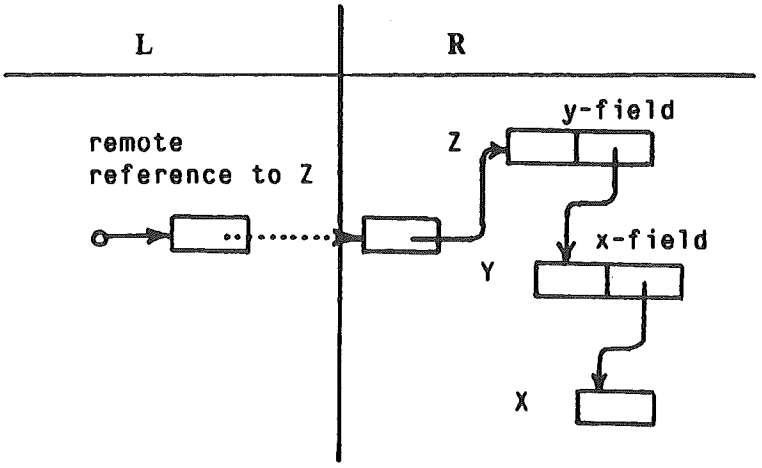
More generally, we may seek to package up sequences of remote executions of code and/or remote data references (when there is no interaction with the intervening code, if any) on the plausible assumption that the extra cost of making a more complex remote transfer or reference is far less than the overheads of performing them separately.

In the case of invoking some function that acts on a large amount of remote data, we may choose to invoke a copy of that function on the remote side as a more expedient optimization than making a local copy of (or floating) a large data structure.

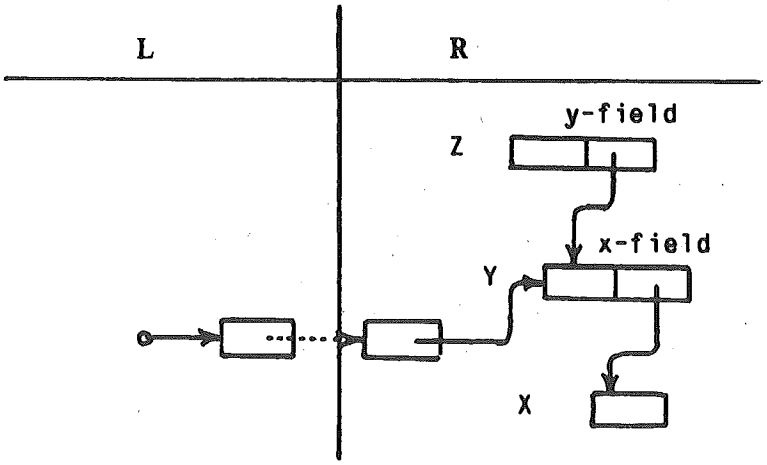
Many other possibilities become available once we consider parallel execution of the partitions; this is one of our longer-term goals.

11.8.3 Run-time Activity Optimizations

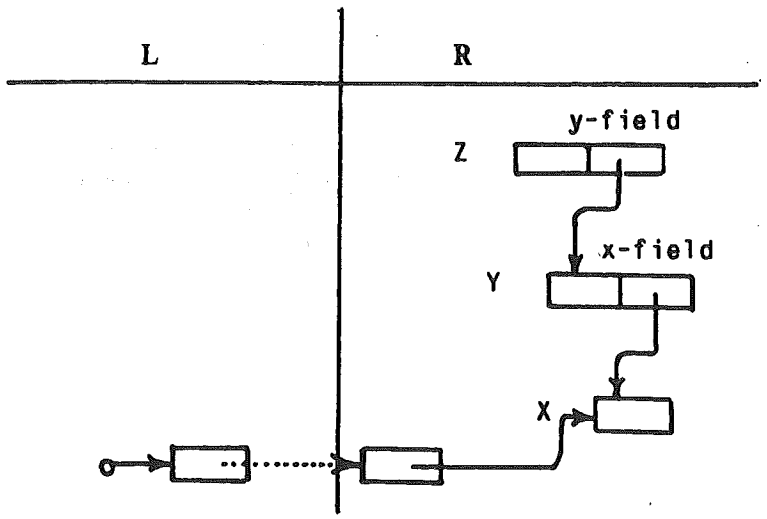
Compile time analysis may reveal unnecessary protocol activity which can therefore be pruned away, giving a guaranteed improvement. Such unnecessary activity often arises because the partitioning system interposes a protocol to cope with the general case. Special case optimizations can be employed to eliminate any unneeded activity. The extent to which such optimizations can be employed will hinge on the depth of analysis performed.



To compute the x-field of (the y-field of Z) where Z and its y-field, Y, are both remote (in side R), the naive protocol would first perform a remote access to get the y-field of Z, returning the value to side L:



Then it would perform a second remote access, to get the x-field of Y:



Clearly it would be an optimization to compute the x-field of (the y-field of Z) in R, eliminating the intermediate crossing from R to L and back.

Figure 11-5: Optimizing nested remote field accesses

Potential savings exist in both the naive protocol and in the results of the preceding sections' optimizations. An example of the former is the passing of values of all global and free variables on each control transfer--flow analysis can lead to a dramatic pruning of such unnecessary passing. An example of the latter is that following the introduction of local copies or floating data, compile-time analysis may reveal occasions where locally valid copies are assuredly present, and direct references to them may be inserted in place of indirection through the protocol interpreter.

11.9 UTILIZING THE APPLICATION DOWNLOADING METHODOLOGY

Although this project has concluded with the development of the prototype partitioning system and optimizations described above, we provide here our thoughts on a support environment for this methodology and further work required to make this methodology practical.

11.9.1 Tools and Instrumentation

A major part of the effort involved in developing the methodology of semiautomatically splitting an existing application program consists of investigating the support required for the programmer. In this section, we discuss the areas in which we expect the Application Downloading system to provide guidance and assistance to the programmer. In particular, we describe the types of support required by a programmer attempting to optimize a partitioned program by applying transformations. Also, we discuss the tools that might assist him in selecting an appropriate partitioning.

11.9.1.1 Transformation technology

Several of the optimizations mentioned above are very context sensitive, in that considerable intelligence is required to decide whether to apply the particular optimizations. Some of these optimizations can be accomplished by declarative information added to the program text, but some will be so local as to require individual program fragments to be optimized. Certainly, optimizations will be interdependent, e.g., collapsed access paths interfere with local copy optimizations.

Hence, an interactive optimization system will be necessary to support the programmer of such applications: he must direct the optimization and the system must maintain a consistent representation of the program and prevent the programmer from making inconsistent optimization decisions. Such systems are presently being developed as "Transformational Implementation" systems [1].

Transformational implementation systems allow the programmer to apply optimizations by choosing individual transformations to be applied to the program--the system carries out the optimization. In order to use this technology, we must be able to characterize the optimizations above as succinct bits of programming knowledge which can be applied by the system when directed by the programmer. We have been active in developing a transformational implementation system and do intend to incorporate it into the application downloading framework in the future.

11.9.1.2 Tools to aid partitioning

A good partitioning is one which limits the number of interpartition transactions--one which transfers control and accesses data across the partition as rarely as possible. This characteristic of a split application relies on two factors: the programmer's selection of a partitioning and his ability to optimize the resulting distributed system. Choosing a suitable partitioning is a difficult task that often depends on the dynamic characteristics of the application program.

There are two classes of assistance that the Application Downloading system could provide the programmer. The first is to present the results of static analysis. Such analysis could produce cross reference listings, tables of the source of references to given types, etc. In addition, static analysis of a chosen partitioning could supply information on the likelihood of frequent interpartition transactions (e.g., remote record access or function call occurring inside a loop) or a table of remote data references (possibly indicating an improper choice for ownership of a given type).

The second class of assistance that could be provided to the programmer includes the results of dynamic analysis. After a partitioning had been produced (and, if desired, optimized), the resulting system could be simulated in a single processor environment. In this way, detailed statistics could be gathered on aspects of the computation such as the source and frequency of interpartition transactions. These statistics could then be analyzed to point out to the programmer the sites of bottlenecks, and possibly suggest modifications to his chosen partitioning in order to improve performance.

11.9.2 Strengthing the Application Downloading Methodology

First, more flexibility must be provided in the choice of a partitioning (for instance, by relieving restrictions on structure and copy types).

Then, methods must be developed for incorporating parallelism into the distributed computation. Many interactive applications lend themselves to such treatment. Semiautomatically constructing such distributed, parallel applications fits in nicely with the existing Application Downloading methodology, since we expect the programmer's interaction to be crucial in this area.

Finally, the ultimate goal of this research is to develop an Application Downloading system which produces distributed applications for execution on real terminals using realistic communications protocols. The system should thus, be redesigned to handle more widely used application languages, such as Pascal or, in the future, Ada.

REFERENCES

1. Balzer, R., N. Goldman, and D. Wile, "On the transformational implementation approach to programming," in *2nd International Conference on Software Engineering*, pp. 337-344, IEEE, October 1976.

2. Christensen, C., and E. Pinson, "Multi-function graphics for a large computer system," in *1967 Fall Joint Computer Conference Proceedings*, pp. 697-711, IFIPS, 1967.

3. Cotton, I. W., "Languages for graphics attention handling," in *Proceedings Computer Graphics 70 Symposium*, Brunel University, 1970.

4. Hamlin, G., Jr., *Configurable Applications for Satellite Graphics*, Ph.D. thesis, University of North Carolina at Chapel Hill, 1975.

5. Kulick, J., *Themis--A Distributed Processor Graphics System*, Ph.D. thesis, University of Pennsylvania, 1972.

6. Stabler, G., *Interconnected Processors--ICOPS*, Ph.D. thesis, Brown University, 1974.

7. Teitelman, W., *Interlisp Reference Manual*, Xerox Palo Alto Research Center, 1978.

12. MULTIAPPLICATION SUPPORT TERMINAL

Research Staff:
Louis Gallenson
Alvin Cooperband

Support Staff:
Lisa Vail

12.1 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.

12.2 GOALS AND APPROACH

The MAST hardware should be low cost, flexible, readily available, and simple, not only from the perspective of the user but also in interaction with responsible applications, system-level software (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.

The applications selected for demonstration are an NLS[6] (On-Line System) work station, an NVT, an NSW (National Software Works) 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 MAST screen editor will provide the user with a simpler, convenient, and more 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 application. It is intended, however, that this division of labor be invisible to the user.

12.2.1 Hardware

During the early phases of the project it was assumed that we at ISI would have to develop the hardware for this terminal since industry was not being very responsive to our needs. During the last year, however, there was strong evidence that the required hardware would be available from reputable manufacturers at reasonable cost, so our hardware effort was terminated. Orders were placed for the Three Rivers Computer PERQ systems on the assumptions that: (1) the cost of obtaining prototype hardware from the vendor would be less than that of building it in house and (2) the relatively expensive PERQs would provide a good source of terminals for a MAST environment if we could reduce the cost by eliminating selected functional capabilities. Unfortunately, the PERQ system has not yet been delivered.

---

[6] NLS was developed at the Stanford Research Institute in the mid-60's and was one of the first office-automation systems.

12.2.2 Editor

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 the kind of document feature he is using, 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 is 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 a wide range of output devices. 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.

The display formatter and the terminal-resident part of the editor 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.)

12.2.3 Application Software Development

A display editor, suitable for use in the documentation system described above, has been developed; it supports a full range of manuscript editing and formatting functions but not the final stages of document generation, e.g., pagination, cross-referencing, front matter, and indexing. An incremental display formatter has also 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. While 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 (see chapter 11. of this report).

12.3 PROGRESS

This project has not been able to obtain the needed hardware. Project members have lost interest and have been busy with other endeavors. Virtually no effort has been expended during the past year.

The MAST editor was reviewed by an ISI group for possible use as a production document-preparation system. The conclusion of this group was that the MAST editor and formatter are worthy of research and should be documented and publicized but are not adaptable to a production system for use at ISI because of the effort required to complete and test a software system, reservations about performance, and the lack of availability of existing editors able to perform the task. Based on this evaluation, the effort of implementing the editor on the PERQ has been temporarily halted. For demonstration purposes the MAST editor and formatter will run on the host machine, and the PERQ will provide the video display and multiple window control functions.

The editor, its intent, approach, and implementation are fully explained in "A Documentation System" [1].

12.4 IMPACT

The major impact of the MAST effort will be noted after successfully integrating and demonstrating our existing software in the PERQ. However, interim results are worth noting. An HP/NLS work station was implemented during the earliest phase of the project. It is based on the HP26xx series, the standard at ISI. It is economical, supported, reasonably packaged, compatible with all other software available to the user, and easily maintained. The HP/NLS work station provides all users with an excellent alternative to the PERQ. ISI has already fabricated some 50 NLS terminals.

A large-screen terminal (54 lines of 80 characters), also developed within this project, has been well received by potential users in the ARPA community. It is compatible with all the existing software at ISI (including NLS and the SIGMA terminal firmware) and the higher character capacity adds a desirable dimension to the terminal/user interface.

The documentation for the editor and terminal has been widely distributed throughout the ARPA research and industry communities. MAST has provided 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.

12.5 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) and as a personalized computer, and integrate the system into ISI's facilities.

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).

13. QPRIM

Research Staff:
Louis Gallenson
Joel Goldberg
Alvin Cooperband

Support Staff:
Lisa Vail

13.1 PROBLEM BEING SOLVED

The QPRIM (QM-1 Programming Research Instrument) effort is producing an online interactive emulation facility housed in an existing mature operating system. Such a facility must be convenient to use for both emulator development and emulator-based target development. It must also be a production system, capable of being acquired and maintained by standard means.

Systems for developing computer programs provide a programmer, or a programming team, with a complete program-development environment that includes a complete set of tools. Unfortunately, the testing and debugging of programs created for embedded systems is typically performed either on the actual machine involved or on an emulator housed in a separate emulation facility. In either case, the program execution environment is different from--and typically far more primitive than--the program development environment. QPRIM aims to bring this program execution and testing into the programmers' working environment without paying the typically prohibitive cost involved in utilizing a simulation program on the development host.

13.2 GOALS AND APPROACH

The PRIM project [2, 5] built such a prototype facility within the TENEX operating system; that facility was operational at ISI from 1974 until 1979. Unfortunately, PRIM suffered from two major defects. First, the emulation host was the Standard Computer Corporation MLP-900 [15]. This particular machine was built as the prototype for a product that Standard eventually abandoned, leaving our MLP-900 a white elephant. Second, since PRIM was designed with speed of emulation as the foremost consideration, the instruction set that emerged was rather idiosyncratic. Consequently, emulators tended to be messy to write and even worse to read. GPM [12], the compiler and language used to generate MLP-900 microcode, helped somewhat to hide the mess. Even so, PRIM emulators were both expensive to create and, due to the uniqueness of PRIM, of limited utility.

QPRIM is a PRIM-like emulation facility running under the DEC TOPS-20 operating system and utilizing a production-emulation engine, the Nanodata QM-1. QPRIM emulators are written in Smite, a computer description language developed by TRW. Under QPRIM, the QM-1 is a new, sharable TOPS-20 resource that is available to TOPS-20 user processes (and, thereby, to TOPS-20 users) to run emulation jobs. Each user process utilizing the QM-1 does so independently of all others; the procedures for constructing and running such an emulation job are quite simple.

13.2.1 System Description

QPRIM was built and runs under the TOPS-20 operating system on a DECSystem-20. QPRIM consists of three major components: the emulation facility per se; a large interface program that gives a user (at a terminal) interactive access to the emulation facility and provides a framework for the running emulator; and the set of target machine emulators that have been built and made available.

The emulation processor in QPRIM is a Nanodata QM-1 [11]. The QM-1 has been produced in modest numbers and is still being manufactured; it has been used for production emulators, research in emulation, and some classroom work. There is, therefore, a community of people who are already producing emulator code for QM-1's. In addition, there has been work on the production of QM-1 microcode from high-level machine descriptions. Currently, there is one language and accompanying compiler, the Smite computer description language [14] produced by TRW. Additionally, an effort to translate from ISPS [1] to Smite is currently under way at ISI.

The QM-1 features two levels of firmware for implementing the user's target (or macro) machine: a low-level 360-bit horizontal "nano" machine and a higher level 18-bit vertical "micro" machine. Much of the potential power of the QM-1 is derived from the ability to tailor the micro machine to the application through the creation of appropriate nanoprograms. Nanodata has defined and implemented a base microinstruction set of some 80 instructions (out of an opcode space of 128 microinstructions) called MULTI [10]. Most applications of the QM-1 use a microinstruction set that is an extension of MULTI, and certain of these extensions have gained considerable distribution.

13.2.2 The Emulation Facility

As an emulation engine the QM-1 is slaved to a DECSystem-20 to become its emulation resource. There are three components that together create this TOPS-20 emulation facility:

- a hardware interface from the QM-1 to the DECSystem-20;
- a TOPS-20 monitor addition, the QM-1 driver, to manage the QM-1 and to provide process access to the QM-1; and
- a firmware emulation-control system resident on the QM-1, the microvisor.

The interface provides the physical connection between the two machines, while the QM-1 driver provides program access to the QM-1 from TOPS-20, and the microvisor supervises emulation job execution to maintain its integrity and that of the TOPS-20 system.

The interface unit that connects the QM-1 to the DECSystem-20 must meet the requirements both of QPRIM, to support the emulation facility, and of general system maintainability. For the latter, it must provide a manual disconnect function that logically separates the QM-1 from the DECSystem-20 and precludes the possibility of one interfering with the operation of the other. Second, to be able to reproduce QPRIM, the interface must be a standard part, available for purchase and subject to normal maintainance. The emulation facility requires that the QM-1 and its microcode have shared access to the memory of the DECSystem-20; all QM-1 memory references to the DECSystem-20 memory must use virtual addresses whose translation to real physical addresses is controlled, ultimately, by the TOPS-20 monitor. In addition to shared memory, there is need for a control communication path between the driver and the microvisor.

After discussions of various options, both internally and with DEC and Nanodata, we selected an interface design in which the memory access path is through an independent port into the DECSystem-20 memory and the control path is routed through the 20's IO Bus. With this design all hardware changes and interfaces lie outside the DECSystem-20 and, therefore, in the QM-1 environment. This arrangement is best because Nanodata seemed far more interested than DEC in participating in this development and following the interface, and the entire concept, to fruition.

The microvisor comprises all the system firmware that runs on the QM-1 to support QPRIM. It consists of nanocode that defines the actual microinstruction set, plus supervisory microcode that defines the operating environment for all QPRIM emulation jobs. The overriding concern in the design of the microvisor is security of the total system--nothing that a user job might do (or attempt to do) can be allowed to gain absolute control of the QM-1 or to gain access to absolute TOPS-20 main store. An opposing concern is that the microvisor should support as rich and flexible an emulation environment as might be useful in the foreseeable future. A final concern is compatibility with other projects that are involved in QM-1 emulation work and the generation of QM-1 microcode. The microvisor design phase went through many iterations and took much longer than expected as the number of competing issues was larger than had been anticipated.

The current microvisor specification [7] provides a fixed microinstruction set, based on Nanodata's MULTI, sufficient to allow the execution of emulators written in the SMITE computer description language. The nanocode to support this instruction set is similar to the nanocode that supports it in standard QM-1 systems with one exception: the main store instructions have been completely reimplemented to treat the main store address as a virtual address within the virtual address space assigned to the emulation job. The main store nanocode also implements, as a dynamic option, automatic reference break detection utilizing the two most significant bits of each main store word as break meta-bits; when this option is active, the executing microcode sees a 34-bit memory. The remainder of the microvisor is microcode that executes on the QM-1 in supervisory mode; the two halves of that code are concerned with emulation job control (including job swapping and page table management) and with intrajob task scheduling in the multitask job environment that is supported by QM-1. An emulation job can be interrupted after any microinstruction, swapped out, and eventually swapped back in transparently; no job status or information is retained in the QM-1.

The driver design provided very few issues. The QM-1 was added as a new device type supported by the TOPS-20 monitor with the driver as the device-specific code supporting that new device type. The driver defines the monitor calls (in terms of existing device-dependent monitor calls, GTJFN, OPENF, CLOSF, GDSTS, BIN, and MTOPR) that are used by a program to create and control a QM-1 emulation job; it also prescribes the format of the main store and context of that emulation job. Main store is a standard TOPS-20 process virtual address space (a fork) created by the program. The context is a contiguous memory area (within the program itself) containing an image of the emulation job's control store; the context is copied into and back out of the real QM-1 control store by the microvisor as part of its job-swapping function. When the job is not running, the context, as well as the main store, may be examined and modified arbitrarily by the controlling program; while the job is running (in the QM-1, of course), only the main store is accessible to the program.

13.2.3 The QM1 Program

The interface program, known as QM1, is that part of QPRIM visible to the user. It is a large interactive command processor that knows how to create and control a QM-1 emulation job on behalf of its (human) user at a terminal. (While the emulation facility is a normal system resource available to

any user process wishing to utilize it, QM1 is the only such program that currently exists; each incarnation of the program creates a process that runs one emulation job, with no knowledge of, or interaction with, any other emulation job.) The interface program is the single part of QPRIM taken directly from the earlier PRIM system. Only a small part of this program is concerned with the details of a QM-1, as opposed to MLP-900, emulation job; by far the greater part of the program--and of the effort invested in writing the program--went into the command processing, the debugger, the emulation job's I/O server, and the management of data and tables concerning all of these.

The change of emulation engines by itself involved only the rewriting of two program modules: one responsible for direct communication with the QM-1 driver to control the emulation job, and the other responsible for loading emulator object code from appropriate object files into (the allocated image of) control memory. These two modules reflect the change from MLP-900 to QM-1 and from GPM (the MLP-900 high-order MOL) to SMITE.

Before making the absolute minimum program change required by the hardware change, we considered what we might do to make QM1 ease the eventual job of the QPRIM emulator writer in preparing emulation jobs. Our assumption here has been that any extra effort on our part at this stage pays a dividend each time a new emulator is added to QPRIM. The areas of concern were the run-time environment that the SMITE compiler presumes for its generated code, the lack of modularity in generation of the emulators, and the possible use of an MPDL-like connection language to describe the configurations of the final emulated computing system (rather than the configuration commands extant in QM1). The scenarios and consequent levels of effort kept growing (see [8] for a flavor of the high-water mark) until we finally realized that we had drastically exceeded the charter of this straightforward engineering project.

In the end, we backed off from most of the improvements, settling on the current manager model that offloads about half of the QM1 debugger support required of a PRIM emulator. The manager is a small, fixed task that QM1 automatically loads along with the emulator being used (by either emulator writer or target programmer). The result is an emulation job consisting of two tasks: the manager as parent and the emulator proper as sole offspring. The manager collects all detected breakpoints in a breakpoint buffer known to the debugger and signals the emulator to stop at the top of its cycle when a break occurs. In addition, reference breaks in the emulator's main store are detected by the QPRIM nanocode and reported directly to the manager for collection; the emulator is not explicitly involved. The emulator itself must still define and detect the occurrence of event breaks; each break need only be reported to the manager. The reporting mechanism has been designed to fit cleanly into SMITE's port construct: each defined event in the emulator consists of a declared flag (for enabling the event) and a port (for reporting an event occurrence) plus a simple conditional statement located at the appropriate point in the emulator code:

```
declare             '' declarations for an event named zzyzx ''
  eflag-zzyzx flag, eport-zzyzx <35:0> port;
  ...
  if eflag-zzyzx    '' at the point defining the event ... ''
   then eport-zzyzx <- {appropriate event parameter} ;
   end if;          ''  we write the port if enabled. ''
```

13.2.4 The Emulators

QPRIM is designed to support a set of emulation tools, letting each be used by its target programmers directly--with no knowledge of QM-1s and very little of QPRIM. In order to integrate an emulator into the QM1 program environment, the emulator writer must follow a set of rules in writing the emulator and must provide necessary syntactic and semantic information to QM1 in the form of a set of tables. The tables allow QM1 to know the names and attributes of all the target machine constructs of interest to the target programmer, while hiding the details of their residence within the address space of a QM-1. The tables are identical in format and content to those used in PRIM, while the rules regarding the behavior of the emulator are very similar (though somewhat simpler because the manager now handles the task of breakpoint logging and reference break detection). When the programmer runs the program associated with his target machine, he gets the QM1 program with his emulator and tables already loaded and ready to go.

For an emulator writer who is developing and testing an emulator, QM1 has a QM-1 pseudo-emulator that consists of tables describing the resources of the QM-1 that are of interest. The emulator writer using this pseudo-emulator has available the full QM1 command processor and its facilities. He runs QM1 directly and commands the program to load his emulator (and tables, should he not want the default pseudo-emulator).

The emulator writer developing a new emulation tool to be installed in QPRIM typically begins with the QM-1 tables. As his emulator begins to take shape and work, he will want to expand the QM-1 tables to include many of the constructs of his target machine. By the time he is done, his tables are approximately the sum of the original QM-1 table and the table required to accompany his emulator for use by others. The final step in the addition of this new emulator is to move (or copy) the emulator code and tables to their final place and create the dummy program that the target programmer will call upon to run QM1.

While we do not intend (or want) to be the principal emulator writers for QPRIM, we feel that it is necessary for us to write and use at least one emulator as part of system shakedown. We have selected the AN/UYK-20 as our target. The UYK-20 emulator in PRIM was both the most comprehensive and the most used. The existing GPM emulator is being hand-translated to MULTI for QPRIM; the resulting emulator should provide a good benchmark of QPRIM vs. PRIM. It may also prove useful in comparing SMITE to hand-coded MULTI.

13.3 PROGRESS

After technical discussions with Nanodata, ISI submitted the final specification for the DECSystem-20 interface unit [3] to Nanodata in August 1978; the prototype unit was installed on the ISI QM-1 in March 1979 and checked out through the following month. Logic errors were found and corrected; most of them proved to be simple mistakes or minor misinterpretations of the specification. Since that time the unit has been in continual (though low-duty cycle) use without problems--a much better record than that of the QM-1 itself.

Soon after the acceptance of the ISI interface unit, the second unit was ordered for RADC. It was delivered to Rome and installed in June 1980, and immediately proved to be totally unusable; virtually nothing in the unit worked correctly initially. As of the end of 1980, Nanodata has made some improvements, but further checkout is still required before all parties are willing to accept it.

Following installation of the ISI interface unit, the pace of development of the emulation facility software--microvisor and driver--was increased. The facility went through TOPS-20 stand-alone checkout in June 1980 and has been running and available at ISIB since then. Bugs have been found and corrected in both microvisor and driver, but only one of them brought the entire TOPS-20 system to a halt. All of the facility software is in place at RADC, awaiting acceptance of the interface unit.

The QM1 program and its accompanying manager task are also completed and working. To date, they have been checked out only with small test microprograms since we have no complete emulators. However, since all the old, complex parts of the program are unchanged from PRIM, they should still be working. The UYK-20 emulator (a hand-translation of the PRIM UYK-20 emulator, written in GPM) is still in progress. The first real exercise of the entire system must await the completion of that emulator.

13.4 IMPACT

We believe that QPRIM will become an important tool for a number of systems and packages that are being developed, in particular the National Software Works (NSW) and the Software Architecture Evaluation Facility (SAEF). Both these systems have a place for an emulation tool that can become part of the offered programming environment. The utility of QPRIM should be further enhanced by the relative ease of production of QPRIM emulators.

13.5 FUTURE WORK

The immediate need regarding QPRIM concerns the completion of the basic system described here. The two primary uncompleted items are the UYK-20 emulator and a manual, similar to the *PRIM Tool Builder's Manual* [4], describing the nature of a QPRIM emulation tool--both the emulator itself and the accompanying tables. Insertion of the UYK-20 tool into QPRIM may indicate further changes in the QM1 program necessitated by the shift from PRIM; it is anticipated that these changes will be small and primarily cosmetic.

If QPRIM finds acceptance in its user community, there are several areas where we see a fairly modest investment leading to greater integration of QPRIM into its larger program development environment. First, intelligent use of the symbolic portion of the SMITE compiler object file (currently ignored) could replace some of the manual effort expended in the creation of an emulator's tables. Second, QPRIM currently lacks a robust source of target code usable across all emulators; some form of retargetable cross-assembler would complement QPRIM nicely. The NASA/McDonnell Douglas Meta Assembler [9] is a possible candidate, though we have no direct experience with that program. A loader in QPRIM that can move such an assembler's object code into the target memory would be useful. Additionally, we note that both the QPRIM debugger and any such assembler require a description of the instruction formats and assembly syntax of the target machine; making a single description suffice for both seems an obvious step.

REFERENCES

1. Barbacci, M. R., G. Barnes, R. Cattell, and D. Siewiorek, *The ISPS Computer Description Language*, Carnegie-Mellon University, Computer Science Department, 1979.

2. Britt, B., A. Cooperband, L. Gallenson, and J. Goldberg, *PRIM System: Overview*, USC/Information Sciences Institute, RR-77-58, March 1977.

3. Gallenson, L., and J. Goldberg, TOPS-20 (TENEX) QM-1 interface specifications, 1978.

4. Gallenson, L., A. Cooperband, and J. Goldberg, *PRIM System: Tool Builder's Manual and User Reference Manual*, USC/Information Sciences Institute, 1978.

5. Goldberg, J., A. Cooperband, and L. Gallenson, "The PRIM system: An alternative architecture for emulator development and use," in *Proceedings, Tenth Annual Workshop on Microprogramming*, ACM SigMicro and IEEE TC-Micro, Niagara Falls, N. Y., October 1977.

6. J. Goldberg, A. Cooperband, and L. Gallenson, "PRIM System: A framework for emulation-based debugging tools," in *Proceedings, 1978 National Computer Conference*, pp. 373-377, American Federation of Information Processing Societies, Anaheim, California, June 1978.

7. Goldberg, J., QPRIM microvisor specification, 1980. MMPE Note # 14 (revision 6)

8. Goldberg, J., QPRIM -- user level, 1980.

9. *Meta Assembler User's Manual*, McDonnell Douglas Astronautics Company, Huntington Beach, Calif., 1975.

10. *MULTI Micromachine Description*, Nanodata Corporation, Buffalo, N.Y., 1976.

11. *QM-1 Hardware Level User's Manual*, third edition, Nanodata Corporation, Buffalo, N.Y., 1979.

12. Oestreicher, D., *A Microprogramming Language for the MLP-900*, USC/Information Sciences Institute, Technical Report RR-73-8, June 1973.

13. *Advanced SMITE Compiler/SASS Interface Control Document*, TRW Defense and Space Systems Group, Redondo Beach, Calif., 1978.

14. *Advanced SMITE Training Manual*, TRW Defense and Space Systems Group, Redondo Beach, Calif., 1979.

15. *MLP-900 Multi-Lingual Processor Principles of Operation*, Standard Computer Corporation, 1970.

14. COMPUTER RESEARCH SUPPORT

Technical Staff:
Dan Lynch
Ray Bates
Dale Chase
Al Cooperband
Phil Crowe
Dick Fiddler
Lou Gallenson
Glen Gauthier
Jim Hurd
Jim Koda
Ray Mason
John Metzger
Bill Moore
Serge Polevitzky
Craig Rogers
Dale Russell
Lynne Sims
Barden Smith
Dennis Smith
Tom Wisniewski
Leo Yamanaka

Support Staff:
Walt Edmison
Chloe Holg
Andrea Ignatowski
Norm Jalbert
Joe Kemp
Keith Miles
Ron Shestokes
Audree Smith
Scott Smith
Debbie Williams
Mike Zonfrillo

14.1 PROBLEMS BEING SOLVED

The TENEX/TOPS-20 project is responsible for providing reliable computer cycles on a 24-hour, 7-day schedule to the ARPANET research and development community. At the same time, the project makes available to ARPANET users the latest upgrades and revisions of hardware and software. The project provides continuous computer center supervision and operation, and a full-time customer-service staff that is responsive to user inquiries. This project supports two computer installations: the larger at ISI's main facility in Marina del Rey, the smaller at the Naval Ocean Systems Center (NOSC) in San Diego.

14.2 GOALS AND APPROACHES

The TENEX/TOPS-20 project provides support in five interrelated though distinct areas: Hardware, System Software, Application Software, Operations, Customer Service. The goals and approaches of each are summarized below.

Hardware

To achieve a reliability goal of 98.7 percent scheduled uptime, the preventive and remedial maintenance responsibilities have been assigned to an in-house computer maintenance group. This group provides 20-hour, 5-day on-site coverage and an on-call standby coverage for after hours. The maintenance philosophy of this group can best be stated as "if it isn't broken, don't fix it." For this approach to be successful, preventive maintenance is very closely controlled, and on-line diagnostics and analysis are emphasized. A primary component in the reliability and availability of the hardware is the physical environment in which it exists. Accordingly, a significant amount of time and resources is expended in insuring that the best, most cost-effective environmental controls are used at both facilities that ISI services.

System Software

The software group's primary goal is to install and maintain, at maximum reliability, ISI's TENEX and TOPS-20 operating systems and applications software. In order to accomplish this goal, the group provides 24-hour, 7-day coverage to analyze system crashes and to provide appropriate fixes. In addition, it is the group's responsibility to install, debug, and modify both the latest monitor versions and the associated subsystems available from the vendor.

Applications Software

This group's primary goal is to provide software that extends and enhances the functionality of the computers for the user community. This involves production of new tools as well as improving and maintaining existing ones.

Operations

The operations staff is responsible for operating the computers and watching over the systems and the environment. At the Marina del Rey facility there is 24-hour, 7-day on-site coverage. At NOSC normally there is coverage from 7:00 am to 4:00 pm daily; special exercises sometimes require extended hours. When a problem occurs, the on-duty staff isolates it and takes appropriate action. On the night and weekend shifts, the operations staff respond directly to user inquiries. Proper training, experience, and familiarity with the environment are especially important.

Customer Service

The customer service group, based in Marina del Rey, provides a two-way communication link between the users and consulting staff. This is accomplished by maintaining a 12-hour, 5-day on-duty staff for prompt problem resolution and rapid information exchange. (The customer-service staff incorporates the User Dedicated Resource project, which was established in 1977 and now operates in parallel with the ISI Computer Research Support project, as well as other computer centers in the ARPANET community.) The group offers introductory training in the use of both the hardware and software tools available on the ISI TENEX/TOPS-20 systems as well as documentation for new users of the ARPANET, and it also assists in the formulation of user training experiments for large, ARPANET-based military experiments, for example, the U.S. Army XVIII Airborne Division Army Data Distribution System at Ft. Bragg, N.C., and the C3/ARPANET Experiment at Headquarters Strategic Air Command at Offutt AFB, Nebr. Appropriate multilevel user documentation is constantly being generated and distributed to individual users, as well as to remote user group liaison personnel. In accordance with IPTO guidelines, the customer-service group provides appropriate accounting data to ARPA.

14.3 PROGRESS

In the past year, ISI has significantly improved in its ability to meet the increasing demand for TENEX/TOPS-20 computer cycles, of users on the ARPANET and at ISI. These improvements fall into the following catagories: Environmental Changes, Hardware Additions, Software Enhancements, and Netloading.

Environmental Changes

Because the reliability of the computer systems was being degraded by environmental problems, specifically, power surges and inadequate air conditioning, ISI upgraded both the power sources and the air-conditioning facilities. Two 150kw motor generator units were installed, thereby eliminating the negative impact of short-term power fluctuations. For the air-conditioning problem, it was necessary not only to correct for past inadequacies, but also to add cooling for the recent 1000-square-foot enlargement of the center. To accomplish this, ISI installed two 20-ton air conditioning units, increasing total capacity to 122 tons for the 4300-square-foot center.

Hardware Additions

During 1980, ISI added a KL-10-based TOPS-20 system, upgraded ISIA from a KA-10 to a KI-10-based system, and upgraded ISIB from a KI-10 to a KL-10-based system. Along with these changes, memory storage was increased by 2.5 megawords and disk storage by 2.6 gigabytes--across six systems. A Model 316 IMP (No. 27) was also added during the year. The Xerox Xerographic Printers (XGPs) at ISI and IPTO were replaced by a laser-light-source, minicomputer-driven, xerographic printer subsystem called the Penguin. The Penguin has consumed a large amount of time and effort, but promises to provide excellent print-out and font versatility. The current local hardware configuration is shown in Figure 14-1. The remote NOSC hardware configuration is shown in Figure 14-2.

System Software Enhancements

The software group converted two systems (ISIB and ISIC) from TENEX to TOPS-20 operating systems. Modifications to the monitor and utility programs to support plug-compatible peripherals were key to this conversion. The many software packages that were improved or developed included:

- Communications. Introduced LTLE (Line at a time - local echo) techniques in support of the Ft. Bragg experiments.
- QM-1. Developed monitor code to support this microprogrammable engine for use by the QPRIM project.
- TCP/IP. Investigated and corrected many operating system bugs uncovered by packages being developed in the Internetwork Concepts Research project.
- Diagnostics. Upgraded SA10 diagnostics to reliably interact with larger system storage configurations. Installed latest versions of vendor-supplied diagnostics as they became available.

Applications Software Enhancements

Most of the progress has been in developing software to drive new print devices, such as the Penguin, and in providing many additional software packages. Some of these include:

- Accounting. Improvements in this area dealt with ease of use and the accuracy of reports.
- PNG. Developed the ability to interface to the Penguin in order to produce high-quality, hard-copy output.
- NLS. Produced a new version for output on the Penguin, improved the debugging capabilities of NLS, and installed a new hyphenation dictionary that tripled the number of known words.
- SCRIBE. Contracted with Scribe's distributor to use the commercially available package, which has the capability to interface with the Penguin.

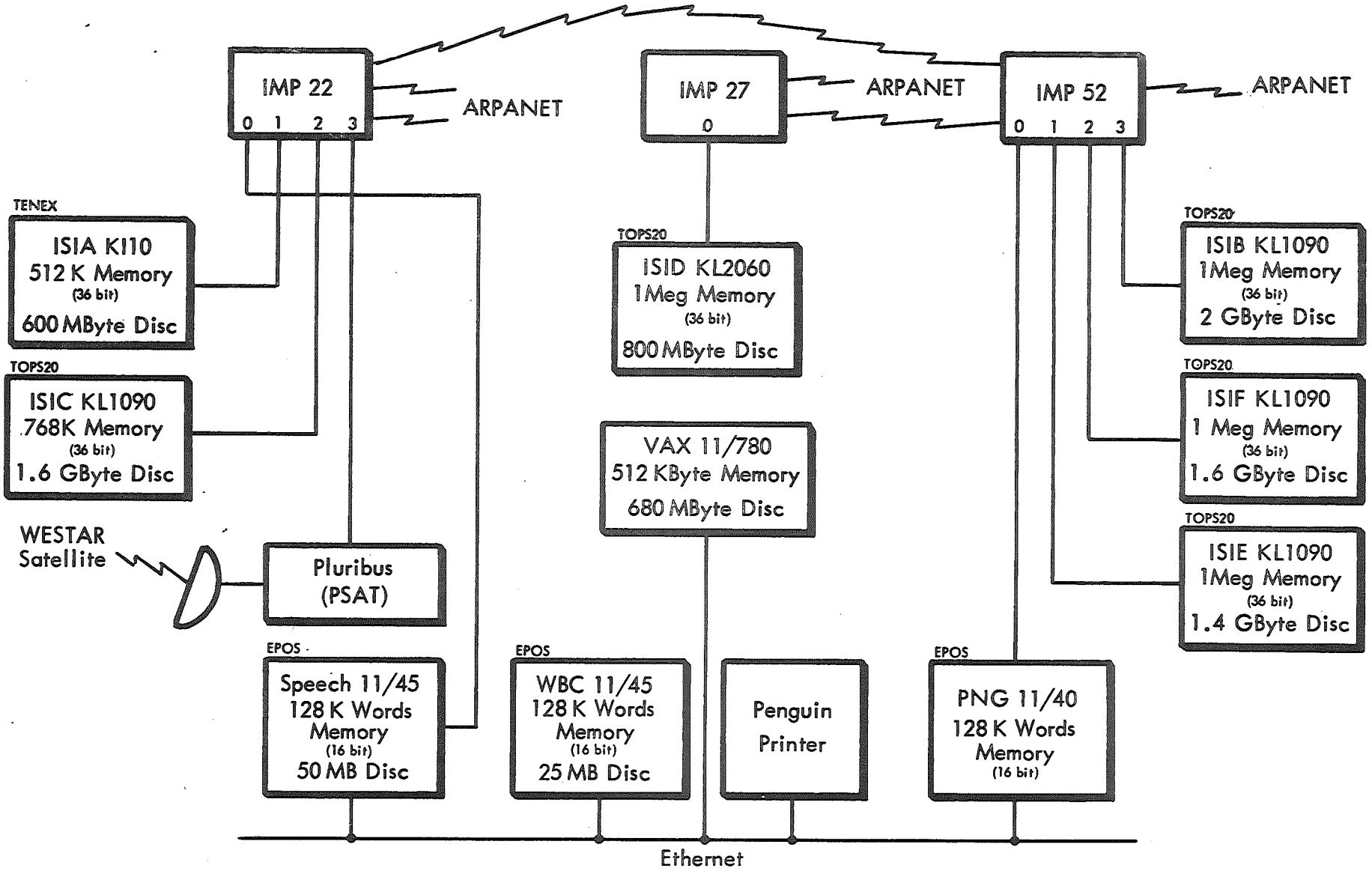


Figure 14-1: Diagram of local ISI ARPANET facility

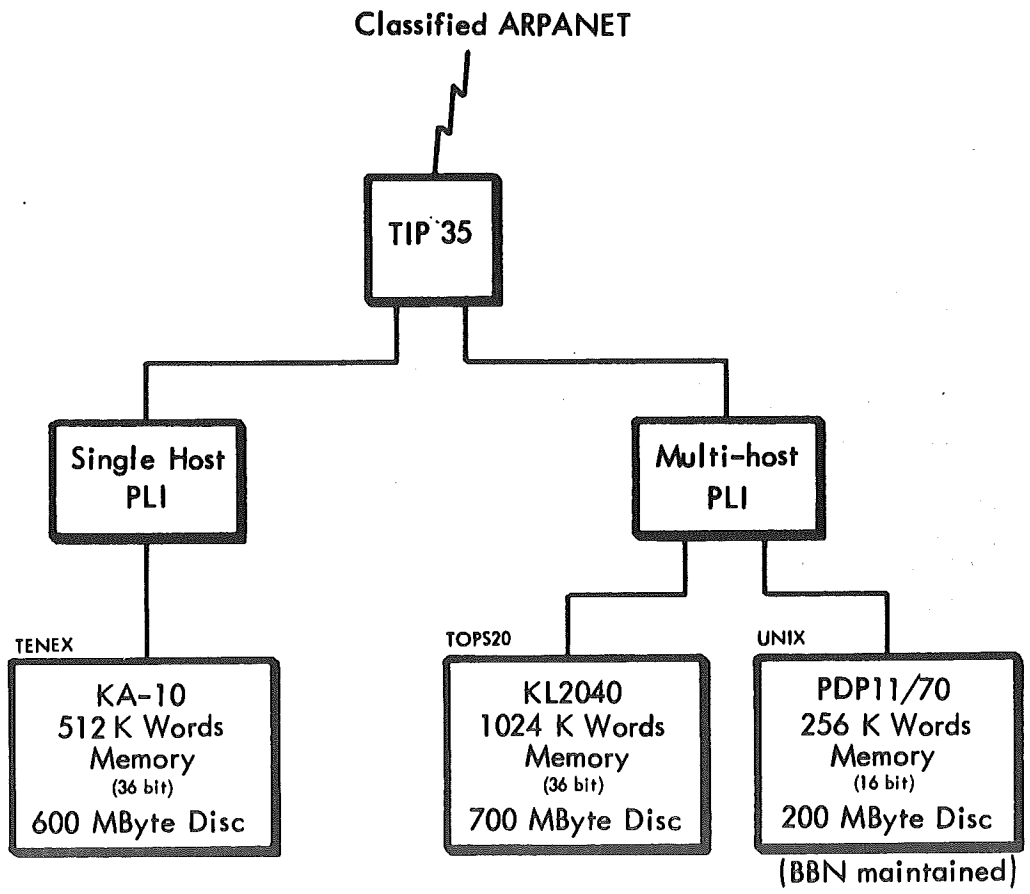


Figure 14-2: Diagram of remote ISI ARPANET facility at NOSC

Netloading

At Ft. Bragg, N. C., some members of the user community responsible for ADDS (Army Data Distribution System) were concerned about very poor response from their host, ISID. At the time of the problem, the ISID population originated from essentially two nodes, Ft. Bragg (40-50 users) and Gunter Air Force Station, Montgomery, Ala. (40-50 users). Since the ARPANET is not optimized for this type of operation, poor response time might have been expected, but there was no justification for a single packet to require round-trip times greater than ten seconds.

A number of tests were conducted to determine the source of the problem. First, ISI checked the response time over the subnet between Ft. Bragg and ISI. BBN (Bolt Beranek and Newman) also measured this response and supported ISI's conclusion: the subnet, though sluggish, did not significantly contribute to the extreme delays. Second, ISI investigated the ISID IMP, IMP 22, as the cause of the slow response. Data from TRACE (a BBN net measurement program) indicated that ISID did not contribute to the very long echo times; rather, an overworked IMP 22 ran out of resources, which resulted in significant delays because of retransmissions. Additional tests showed excessive waits for packets sent to ISID. ISI examined usage of NCP buffers on ISID and found zero buffers available for significant percentages of time. The available number of NCP buffers was doubled and locked in core to provide faster access. Tests were repeated for several days showing much improved results. There were very few ten-second responses, and the few recorded were correlated with subnet problems. However, the response time during peak activity remained poor.

The TOPS-20 NCP program was then modified to cooperate with the IMP protocols that cause the IMP to stop talking to a host (in this case ISID) when more than eight outstanding messages are aimed at a single destination (such as Gunter TIP). This significant change showed that the jerky response tc Ft. Bragg could be attributed to overloading the Gunter TIP and the resultant blocking of messages tc the Ft. Bragg TIP until the Gunter TIP became unclogged. With the new, nonblocking NCP on ISID (and all other TOPS-20 sites at ISI), this problem no longer exists.

14.4 MILITARY IMPACT

ISI's computer centers provide ARPANET cycles 24 hours a day, 7 days a week to the Strategic Air Command, Gunter AFS, Naval Ocean Systems Center, and Ft. Bragg. In addition to supplying machine time, the TENEX/TOPS-20 project has provided additional support in the following areas:

- Training, documentation, and modifications as requested by user groups for NLS.
- Planning support and training in ISI systems software for the installation of an on-site DEC System 2060 at Gunter AFB.
- Support for the production of AFM 67-1 with NLS.

14.5 FUTURE WORK

The Computer Research Support project will continue to provide computing service to the ARPA research community, provide and support software packages for the ARPANET community, and offer a program of technology transfer through the customer-service group.

The project also has some specific plans for the coming year. The DEC KL machines that currently run version 3A of the TOPS-20 monitor will be upgraded to version 4. The project will investigate the design and implementation of a computing center system structured so that users' access to their files is not dependent on a single machine. Users should then be able to have access to the resources they are used to having even though a particular computer has failed or has been taken offline for maintenance.

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.

---, *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.

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

---, 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, *On the Use of Programming Knowledge*, ISI/RR-77-63, October 1977.

---, Neil M. Goldman, and David Wile, *Meta-Evaluation as a Tool for Program Understanding*, ISI/RR-78-69, January 1978.

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

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

---, and Dennis Hollingworth, *A Distributable, Display-Device-Independent Vector Graphics System for Command and Control*, ISI/RR-80-87, July 1980.

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.

---, *The Oceanview Tales*,ISI/RR-79-83, February 1980.

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.

---, and David S. Wile, *A Database Foundation for Process Specifications*, ISI/RR-80-84, January 1980.

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.

---, 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 Armar A. Archbold, *Working Papers in Dialogue Modeling*, Volume I, ISI/RR-77-55, January 1977.

---, and Neil M. Goldman, *Process Models of Reference in Context*, ISI/RR-78-72, October 1978.

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

---, *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.

---, *Toward a Speech Act Theory for Natural Language Processing*, ISI/RR-79-75, March 1980.

---, *Dialogue Games*, ISI/RR-79-77, November 1979.

---, *Computer as Author--Results and Prospects*, ISI/RR-79-82, January 1980.

---, 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.

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

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

1979 Annual Technical Report: A Research Program in Computer Technology, October 1978-September 1979, ISI/SR-80-17, June 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.

Bisbey, Richard II, Dennis Hollingworth, and Benjamin Britt, *Graphics Language (Version 2.1)*, ISI/TM-80-18, July 1980.

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