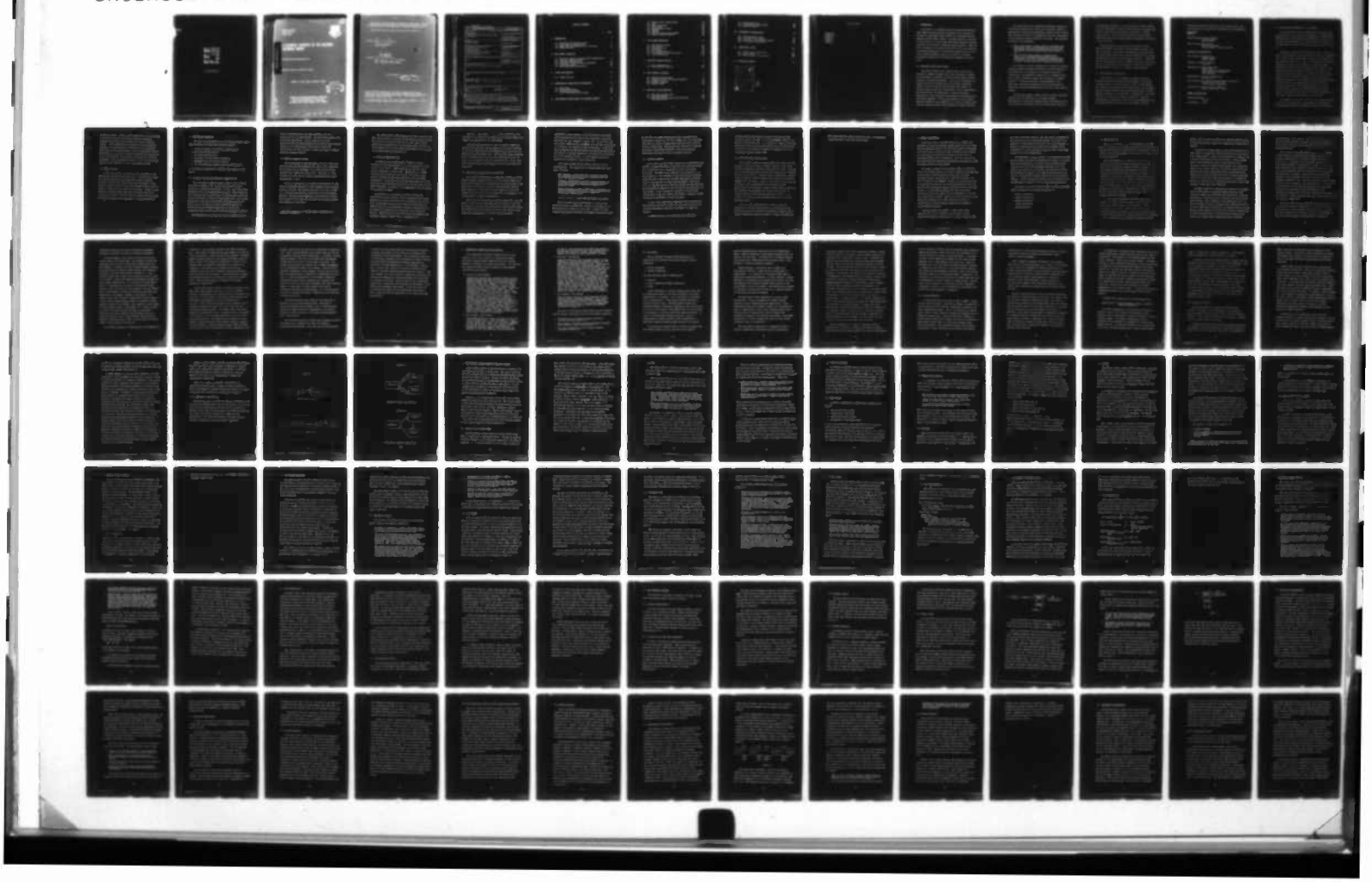
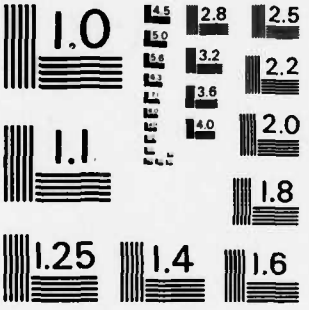
AD-A132 320 A TECHNICAL OVERVIEW OF THE NATIONAL SOFTWARE WORKS(U) 1/2
BOLT BERANEK AND NEWMAN INC CAMBRIDGE MA
R E SCHANTZ ET AL. MAR 83 BBN-5238 RADC-TR-83-80
UNCLASSIFIED F30602-81-C-0213 F/G 9/2 NL






MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS - 1963 - A

RADC-TR-83-80
Technical Report
March 1983



# A TECHNICAL OVERVIEW OF THE NATIONAL SOFTWARE WORKS

Bolt Beranek and Newman, Inc.

Richard E. Schantz and Robert H. Thomas



ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-83-80 has been reviewed and is approved for publication.

APPROVED: Patricia J. Baskinger

PATRICIA J. BASKINGER
Project Engineer

APPROVED:

JOHN J. MARCINIAK, Colonel, USAF
Chief, Command and Control Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COTD) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER<br>RADC-TR-83-80 | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle)<br>A TECHNICAL OVERVIEW OF THE NATIONAL SOFTWARE WORKS PROJECT | | 5. TYPE OF REPORT & PERIOD COVERED<br>Technical Report<br>6. PERFORMING ORG. REPORT NUMBER<br>BBN Report No. 5238 |
| 7. AUTHOR(s)<br>Richard E. Schantz<br>Robert H. Thomas | | 8. CONTRACT OR GRANT NUMBER(s)<br>F30602-81-C-0213 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>Bolt Beranek and Newman, Inc.<br>10 Moulton Street<br>Cambridge MA 02238 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS<br>63728F - 64740F<br>25310115 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS<br>Rome Air Development Center (COTD)<br>Griffiss AFB NY 13441 | | 12. REPORT DATE<br>March 1983<br>13. NUMBER OF PAGES<br>130 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)<br>Same | | 15. SECURITY CLASS. (of this report)<br>UNCLASSIFIED<br>15a. DECLASSIFICATION/DOWNGRADING SCHEDULE<br>N/A |

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

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

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Patricia J. Baskinger (COTD)

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

National Software Works
NSW
Network Operating System
Distributed Systems
ARPA Network

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

This report presents a technical overview of the National Software Works System architecture and design. The NSW is a working example of a network operating system, which is intended to integrate and provide uniform access to software tools residing on a number of constituent host systems connected to the ARPA Network.

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

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

# LIST OF FIGURES

# 1. INTRODUCTION

This report is a technical review of the National Software Works (NSW) project and system development effort prepared for the Rome air Development Center (RADC) and the Defense Advanced Research projects Agency (DARPA). The NSW project was initiated in 1974 as part of a larger ARPA/IPTO program addressing problems concerning the high cost of producing quality software. RADC has been a co-sponsor of the effort since 1975, and is now organizing a technology demonstration of NSW system capabilities for the Air Force Logistics Command, as part of an effort to introduce networking concepts into the operational components of the Air Force.

## 1.1 Background and Technical Need

The National Software Works (NSW) project began with the objective of applying computer networking technology to improve the distribution of software tools and reusable software modules throughout the Defense Department. From the very beginning, the program has had two complimentary but relatively independent thrusts. The first has been the refinement and transfer of existing Arpanet technology to operational service organizations. The other has been the development of resource management techniques for large networks of heterogeneous and geographically distributed computers. The term heterogeneous in this context means the computers can be supplied by a variety of vendors, have different instruction sets, and run different operating systems.

The initial goal of the technology development component of the National Software Works project was to provide a network of geographically distributed heterogeneous computers with a uniform user interface that made the incompatibilities and idiosyncrasies of the individual systems invisible to the users. This approach was based on two hypotheses:

1. That a large number of useful software development and maintenance tools already existed, but were not widely used because they were scattered across a variety of incompatible operating systems making them difficult to use.

2. That it would be much easier to integrate tools together using standard network protocols than to recode them or move them to different operating system environments, and that a convenient and coherent user interface could be achieved with appropriate protocols.

The attempt to retrofit coherence onto a network of existing computers, operating systems and software tools has resulted in the exploration of an important set of system architecture alternatives, and substantial insight into the tradeoffs which must be made in designing higher level protocols for computer networks. It has also resulted in the development of an operational prototype NSW system which is now being used for conducting a technology demonstration for the Air Force Logistics Command.

The idea that computer networks should be used to distribute software tools and expertise is now widely accepted. Publications on the National Software Works were among the first articulations of that idea, and have certainly been a factor in

its widespread acceptance. All the operational network software factories implemented to date have been simple star networks with a single large time-sharing system at the center. The NSW project was one of the earliest explorations of the concepts associated with offering decentralized, heterogeneous services as of such a facility.

The NSW project was also one of the earliest and most ambitious attempts at building a network operating system. The need for systems of this type is evidenced by the proliferation of projects with similar goals, and the coverage given to network operating system issues in the technical literature. Experience with the NSW system has served to identify and focus attention on many of the problems in this new field of research, as well as provide adequate solutions to a number of these problems.

## 1.2 Participating Organizations

Over the years of its existence, a number of different organizations and individuals have participated in the project and helped shape the directions that it took. The NSW project was organized around committees of various types, and the design process required consensus among participating groups of widely differing orientation and background. Consequently, it is often difficult to attribute key ideas or approaches to specific organizations or individuals. With only a few exceptions we will not attempt these attributions. We will however, list the

various participating organizations, their generic role, and a very few of the key individuals contributing to the successes of this project.

Sponsors

Advanced Research Projects Agency:
William Carlson,
Steve Crocker.

Rome Air Development Center:
Patricia Baskinger,
Richard Metzger, Richard Robinson.

Designers and Implementers

Bolt Beranek and Newman Inc.:
Richard Schantz,
Robert Thomas

Honeywell Information Systems:
John Ata

Massachusetts Computer Associates:
Ross Faneuf,
Robert Millstein, Charles Muntz,
Kirk Sattley, Stu Schaffner,
Steve Warshall.

Massachusetts Institute of Technology:
Douglas Wells

SRI International (formerly Stanford Research Institute):
Charles Irby, Jon Postel,
Richard Watson, James White

University of California at Los Angeles:
Robert Braden, Neil Ludlam

Support Organizations

System Operators:
GSG Inc.

Tool Manager: IITRI

## 1.3 Objectives and Perspective for the Report

At various times, different organizations have had overall responsibility for the design and enhancement of the NSW system. BBN was not one of the original participating contractors, but has been largely responsible for recent design enhancements. As a consequence, the task of writing this overview report falls to us.

We have two objectives in writing this report. First, we want to document some of the good ideas and innovative approaches that are part of NSW system design. NSW was quite possibly the most ambitious effort of its time in the distributed system area. Integrating the many diverse aspects of a complete system required that a wide variety of system issues, integral to any distributed system architecture, be faced. This report is a vehicle for documenting some of our solutions to those issues. Second, we want to consolidate under one cover, all of the technical dimensions of the problem addressed by the NSW project. It has been said that system building is one of the more difficult activities in our profession because of the wide diversity of the components of any potentially operational system. This report should serve to indicate the magnitude of the effort needed for future endeavors of this type.

The emphasis of the report will be on the technical aspects of the current system design. Other aspects of the project, including organizational and managerial ones, have at times had

significant impact. However, these aspects will not be the focus of this report and will be ignored except for occasional references where they directly impact the current technical product. Little emphasis will be placed on the chronological changes in the system design in deference to a more complete description of the current design. We will occasionally discuss functionality in its earlier forms to highlight the advantages of the current approach. The perspective regarding the technical assessment of what is important and what is not, as well as what worked out well and what did not, is the author's alone.

## 1.4 Report Overview

In the next section, we provide a brief chronology of the NSW project highlighting its many "eras". Following that, there are sections describing the system and the major technical results of the project. These sections are organized to first provide a system overview, and to then discuss in depth each of a number of different major aspects of the system. A number of the important technical issues which have been raised during the course of the project, and will likely be faced by other projects with similar objectives are discussed in a concluding section.

## 2. NSW PROJECT CHRONOLOGY

The design and implementation of the National Software Works has proceeded in six sometimes overlapping phases:

1. Structural design and feasibility demonstration
2. Detailed component design
3. Prototype implementation
4. Reliability and performance improvement
5. Management Plan and Production System
6. Air Force Logistics Command Technology Demonstration

In the following subsections we describe these phases in more detail.

### 2.1 Structural Design and Feasibility Demonstration

The first phase of NSW development began in July 1974 and concluded in November 1975. During this period, the basic architecture of NSW was established. Further, relatively ad hoc implementations of major components were made. These components were integrated into a system which was demonstrated to ARPA and Air Force personnel at Gunter AFB in November 1975. This system demonstration exhibited various system functions, the use of batch tools on the IBM 360 and Burroughs B4700, the use of interactive tools on TENEX, transparent file motion and translation, and a primitive set of project management functions.

The demonstration confirmed that the desired NSW facilities

could be implemented and that semi-transparent use of a distributed tool kit was feasible. The NSW System, however, was inefficient and fragile. Further, many of the ad hoc implementations had design weaknesses which limited their general application to a sufficiently broad range of hosts and capabilities. For these reasons, an effort was begun to produce effective component designs.

## 2.2 Detailed Component Design

This second phase of NSW development started in June 1975. Specifications were developed for Tool Bearing Host components -- MSG, Foreman, and File Package.[1] All of these specification documents were completed by March 1976. (They have all been revised since then, but the original specifications are still substantially in use.)

During the same period, the external specification for the resource management and control system component (Works Manager) was also produced. The remaining portions of the core of NSW -- i.e., the batch tool facility, consisting of the Works Manager Operator, Interactive Batch Specifier, and Interface Protocol -- were designed during phase one, and those designs were retained until phase four (see below).

---

[1] The major components of the NSW system architecture are presented in Section 3.

The remaining major NSW component, the user interface Front End, was the subject of several design efforts over the course of the project with different features in each. Two of these designs were implemented and are available today, with one of them targeted for the recent system enhancements and to be used for the NSW technology demonstration.

## 2.3 Prototype Implementation

As specification documents were completed, various contractors began implementation of the NSW components on the initial set of hosts -- TENEX, MULTICS, and IBM 360; these efforts commenced in January 1976. Implementation on TENEX proceeded more quickly than the efforts on the other hosts -- primarily because the MSG system designers were also TENEX implementors. By October 1976 prototype implementations, which conformed to the published specifications, had been made for all TENEX components. In addition, all components of the core system were available on TENEX.

Implementation of components on MULTICS and IBM 360 proceeded more slowly; however, initial implementations of MSG components on both of these hosts were completed by the end of 1976. By November 1976 sufficient progress had been made on implementation of a File Package and Foreman on MULTICS that it was possible to demonstrate an interactive tool running on MULTICS. Implementation of 360 (interactive) Tool Bearing Host components reached a similar state in September 1977.

Also during this phase, a Front End which provides a user interface to the NSW functions supported by the Works Manager and Foreman was implemented to run under TENEX.

Prototype implementations of the core system, TENEX tool bearing host components, and the TENEX Front End that conformed to the design specifications were demonstrated to Air Force and ARPA personnel in November 1976. That prototype system supported access to TENEX interactive tools and IBM 360 batch tools, as well as to a rudimentary Multics interactive tool. At the same time, a demonstration of MSG components on all three hosts was also given.

## 2.4 Reliability and Performance Improvement

Even though implementation of components on MULTICS and IBM 360 was lagging, implementation of the core system, TENEX Tool Bearing Host components, and TENEX Front End had proceeded to the point that the issues of reliability and performance assumed major importance. The system exhibited sufficient functional capability that it could clearly support use by programmers if it were sufficiently robust and responsive.

The first task attacked was to provide robustness. Work had begun in 1975 on a full-scale NSW Reliability Plan -- this was the design for multiple Works Managers with duplicate data bases. This detailed Plan was released in January 1977. Since it was clear that implementation of the full Plan was a major

undertaking, a less ambitious Interim Reliability Plan which ensured against loss of a user's files was begun in mid-1976. This Plan was also released in January 1977. By June 1977 the core system, TENEX Foreman, and TENEX Front End had been modified to incorporate the features of that Interim Plan. In addition, both the MULTICS and IBM 360 Foremen (only partially implemented) were altered to conform externally to the scenarios specified by the Interim Reliability Plan. A system exhibiting the new scenarios was released for use in June 1977.

Performance of NSW had been limited from the initial implementation. The reasons for its limited performance were many, including:

- NSW components (which constitute an operating system) were executed exclusively as user processes under the local host operating systems.
- Component implementation had been oriented towards ease of debugging and other concerns of prototype systems rather than towards the performance expected of a production system.
- Strict adherence to logical boundaries in developing the component implementations led to a system implementation which relied heavily on relatively expensive interprocess communication.

In 1977, efforts to improve NSW performance were begun.

The first effort was the development of a performance measuring package for TENEX MSG. Results of the first set of measurements were reported in April 1977. A number of more sophisticated measuring packages were completed by February 1978.

By May 1978, all TENEX components had been instrumented and measurements of page use, CPU time, elapsed time, etc., had been taken under a variety of system load conditions and on several different TENEX hosts. Efforts were then undertaken to develop performance improvements suggested by these measurements. Performance improvement is still an ongoing activity.

## 2.5 Production System

As the effort to improve NSW reliability and performance was underway, efforts were begun to make NSW a more packaged product and a system which could be operated by other than system developers. The TENEX system components were converted to run under TOPS-20 to take advantage of more cost-effective hardware technology and a commercially available base operating system. Tools to help operate the system were developed. Regression tests for the NSW functions accessible through the user interface were developed and applied to each version of the system prior to its release. A user's manual for the system was published. Documentation of the core system was produced. Finally, a draft configuration management plan was developed.

In late 1978 an NSW Product Management Plan was generated. This document identified a number of roles associated with the continued development, operation, and support for NSW as a product.

A computerized tool for coordinating the reporting,

checking, fixing, and testing of software problems has been developed and put into use: it is called MONSTR -- MONitor for Software Trouble Reporting. It is driven by a table of protocols defining the desired interactions between all the organizations listed above, and handles the passage of messages through the appropriate channels between them.

## 2.6 AFLC Technology Demonstration

The Air Force Logistics Command (AFLC) was brought into the program in 1978, with plans to conduct a series of technology evaluation experiments during the 1981-1983 time frame. The plan for the AFLC NSW experiments involves the installation of ARPANET nodes at three major Air Logistics Centers, experimental ARPANET access to mission relevant software tools available on selected ARPANET host, and finally, access to these same tools through the NSW system, using an NSW access host available at each participating Air Force base. As part of this plan, an NSW Front End component was implemented for the DEC PDP-11 family of computers using the UNIX operating system as a base. It is intended that this be the primary NSW access host for the AFLC NSW experiments.

As part of the preparation for the AFLC technology demonstration, the NSW system design and implementation has undergone a thorough evaluation and enhancement phase to add or augment capabilities which were needed to support the expected

AFLC usage patterns. These enhancements have been completed continuing under current RADC sponsorship.

## 3. SYSTEM ARCHITECTURE

The original goal of the NSW project was to design and implement a software development support system which would both provide convenient access to off-the-shelf software development tools which run on a variety of hosts connected to the ARPANET, and also serve as a vehicle for providing various forms of automated software project management. Over time these goals have been clarified, extended and modified as the project evolved and passed through its many phases.

This report focuses on the technical aspects of what has become the dominant theme of the project: designing, implementing and operating a network operating system which provides convenient access to software tools dispersed throughout the hosts connected to the ARPANET. Access to and interoperability of such tools has been one of the principal NSW goals, and has been the motivating factor behind a substantial part of the NSW design. At times there have been other goals which have influenced parts of the system that were developed at the time these goals were prevalent. However, we will not focus on any of these other aspects of the system, preferring instead to concentrate on technical details concerned with building a network operating system.

Since the system has changed, in some cases rather substantially, over its lifetime, and continues to have unimplemented features, describing it is often a blend of what

was, what is and what might be. Our goal here is to present an accurate picture of the current NSW system. Only where it is important to understanding the utility of a current concept, or because a function has been included only to support future enhancement will we go into detail about what was or what might be part of the future NSW systems.

The NSW project is still ongoing, in its demonstration phase and the system continues to be maintained in support of the AFLC Technology Demonstration. The AFLC Technology Demonstration represents the first non-experimental sustained use of the system. As a result, it is too early to evaluate the overall effectiveness of a system of this type as matched against the needs of a typical user community like the Air Force Logistics Command. A more comprehensive evaluation of the functionality of the NSW should be planned to follow technology demonstration.

The remaining sections describe aspects of the current NSW system from each of the following perspectives:

- system architecture
- system functionality
- system reliability
- system performance
- system operation

## 3.1 System Overview

There are two distinct but related aspects of the NSW system. One aspect is the system architecture while the other is the system functionality supported by the architecture. There is often a fuzzy boundary between the two as will be evident in the following discussion.

The basic problem to be confronted by the NSW project was to transform an existing environment consisting of a number of heterogeneous autonomously operated computer systems (including their operating systems and the programs they support) interconnected by a medium speed, connection oriented communication subsystem (the ARPANET and the standard NCP host-to-host protocol) into a more coherent collection of computer services provided to the user under the auspices of a single network operating system. Basic to this objective were key concepts of uniformity and interoperability. Thus we can view NSW as an operating system designed to support users in a computer network environment.

In some respects NSW appears like most modern operating systems. For example, NSW must support common system functions such as user authentication, a filing system, a command interpreter, and so forth. However, NSW differs from conventional operating systems in some very significant ways. The basic building blocks for the NSW system are not traditional hardware components such as processors and memory devices.

Rather, they are existing conventional operating systems and the services they provide, along with an appropriate interconnection medium.

A premise of the NSW concept is that there currently exist a wide variety of application software services which have proven their effectiveness. A goal of the NSW system is to augment the utility of these services by providing users a uniform access path to them regardless of their originating host and to facilitate the use of groups of them together in an integrated fashion. As a result, users can be offered a wider variety of services than by any conventional system, the potential user community for a given software service can be greatly expanded, and new services can be tested and evaluated in a convenient fashion by a diverse set of interested users prior to being made available to the entire user community.

The NSW concept also recognizes that some of the available services may already be reasonably complete systems. NSW represents a means for integrating such systems into a coherent framework for collaboration among individual users and between organizations. Such a framework is made possible by, but not directly supported through, computer network technology and its low level communication protocols. As a network operating system NSW can greatly amplify the utility of the network technology by providing uniform access to, and centralized uniform access control for, objects (data, computing services, programs)

distributed around the network. Without such a utility, network users are often forced into awkward and tedious work patterns which inhibit cooperation and often preclude the use of new software services.

The evolution of the NSW system structure has been shaped by a fairly complex relationship between design, the notion of prototype implementation, and organizational convenience. The structure is a consequence of the earliest decisions to functionally decompose the design into a number of basic units which reflected various aspects of the NSW functionality, and which could accommodate the most complex pattern of activities anticipated for the system. There would be a Works Manager component which represented a logically centralized resource management component. There would be a Front End component which managed the user interface to the NSW functionality. There would also be Tool and File Bearing Host Components for integrating programs and file systems into the NSW concept of host transparent program execution and user transparent file movement and conversion. These components were known as the Foreman and File Package, respectively.

Each active NSW user has a dedicated Front End process which acts as his interface to the NSW system. The Front End acts principally as a command language interpreter making requests upon other components as necessary to satisfy user commands. The Front End process supports a standard NSW user interface,

including a standard set of control functions and commands, regardless of the host on which it is actually implemented.

The Works Manager is the resource allocation and access control module for the NSW system. All attempts to access NSW resources, such as tools or files, must be authorized by the Works Manager. To perform its task, the Works Manager maintains data bases such as an NSW file system catalog, tool descriptor information, and user authentication information. It also maintains lists of the rights and privileges of each user known to the system. Interactions between Works Manager processes and other system components occur on a transaction oriented basis. That is, the system does not dedicate a single Works Manager process to each active user for the duration of the user session. Rather, Works Manager processes are dynamically allocated (and deallocated) as necessary to support a user session. For example, when a user initiates a command that requires access to an NSW resource a Works Manager process is allocated to handle requests related to that command. Upon completion of the command the Works Manager process is deallocated (i.e., either returned to a pool of free Works Manager processes or terminated). Continuity across such instances of Works Manager service is achieved through the use of a shared dynamic data base which depicts the momentary state of NSW, including lists of currently logged in users and their active tools.

File Package processes are responsible for file movement and

translation. A File Package resides on every NSW Tool Bearing Host. Once access to an NSW file has been granted, it is the job of the File Package to make a suitable copy available. The Works Manager process arranges for File Package processes at the file source ("donor" File Package) and destination ("receiver" File Package) hosts to cooperate to accomplish this movement and translation. The receiving File Package drives the copy procedure and has the task of creating a copy with equivalent logical structure as the original. Like Works Manager processes, File Package processes are allocated on a transaction oriented basis. When the file movement is completed, File Package processes are deallocated.

The Tool Bearing Host Foreman is the tool's interface to the NSW. When a user requests the start of a tool, an Foreman process on the appropriate Tool Bearing Host is allocated for the duration of the tool session. The Foreman process provides the NSW execution environment for the tool and controls its operation. This execution environment differs somewhat from the standard environment provided to the tool by the local Tool Bearing Host operating system. For example, when a "file open" operation is initiated by a tool, the operation must be processed in the context of the entire NSW rather than that of the local host operating system. The Foreman process responds to such an attempt by interacting with a Works Manager process to complete the file reference. The Works Manager process consults the NSW file catalog to verify the existence of the file specified by the

Foreman, and that the user and tool are authorized to access the file. Next, the Works Manager acts to ensure that the file can be physically accessed by the Foreman/tool. In general, this may require movement of the file to the Foreman host and possible translation of file data to a form usable by the tool. The Foreman provides each tool instance with a temporary workspace for file manipulation during a tool session. Once the file is physically accessible in the workspace, the tool uses the local host file system primitive operations for manipulating the file data. It is the responsibility of the Foreman to manage the available Tool Bearing Host workspaces and to maintain the isolation of the tool from other processing on the host operating system. In some cases a Foreman process directly uses information provided by the Works Manager (i.e. file descriptors cataloged in the NSW file system) to complete the file reference on behalf of the tool.

Interactions between these components are sufficient to describe all of the basic NSW operations. This type of functional breakdown had the property that it could uniformly describe the system operations independently of the physical location of either the user or the objects being manipulated, e.g., tools or files.

This interaction model of the NSW system components requires an architecture or framework into which the component interactions can be placed. As we shall see, this architecture

is based on the envisioned high level functionality which was originally anticipated, but now takes on an existence quite independent of that functionality. The architecture referred to in this context represents the link between the highest levels of the system design as embodied by processes supporting the functions of the NSW components (e.g., Works Manager, Foreman, etc.), and the lower levels of the system as embodied by the disparate individual operating systems and their resources. This link includes, but is not limited to, system wide conventions for interprocess communication, process management and interprocess and transaction protocols. The major part of the interprocess communication architecture was the design and development of the communication subsystem known as MSG. In the next section we will describe key aspects of the interprocess architecture, before continuing with the description of the higher level NSW functionality.

## 4. INTERPROCESS COMMUNICATION ARCHITECTURE

The interprocess architecture design grew out of the analysis of the various patterns of communication among previously mentioned system components which were thought necessary to implement the system functionality. The following were typical of those original analyses of intercomponent communication requirements.

- Front End - Works Manager

  Communication between these processes consists of user requests for NSW resources (Front End to Works Manager) and Works Manager responses to such requests (Works Manager to Front End). Examples of such requests are: run a tool, copy a file, delete a file, etc. These requests are relatively infrequent - a user may make only a few per hour. Each request is short - almost all requests can easily be encoded in 1000 bits. The response to each request is also short - again, less than 1000 bits. The time required to process a request is generally brief - certainly on the order of milliseconds as compared to the minutes between requests. There is no necessity for a request to be processed by the same Works Manager process that processed any previous request (since all instances of the Works Manager share the same common data base). Hence a communication link need not be retained between a Front End and a Works Manager between resource requests. Thus we can characterize Front End - Works Manager communication as a sequence of unrelated elements, where each element is a short request, a brief delay, and a short response, and there is a long delay until the next element of the sequence.

- Tool/Foreman - Works Manager

  These communications are exactly analogous to Front End - Works communications. A tool (on behalf of a user) requests an NSW resource of the Works Manager. Examples of such requests are: open a file, create a subsidiary tool process, deliver a file, etc. As above, these requests are generally less than 1000 bits, are processed by the Works Manager in milliseconds, have responses of less than 1000 bits, and are relatively

infrequent. The only difference between this pattern and the preceding pattern is that tool requests are more frequent than Front End requests, although the time between such requests is still measurable in minutes.

- Front End - Tool/Foreman

  Communication between these processes consists of user commands to tools and tool responses to users. In some cases these communications will fit into the same pattern as the the three previous cases. Often, however the pattern is different. Consecutive requests are related and must be serviced by the same tool. The time between the user's command and the tool's response may be greater than the time between the response to the previous command and the issuing of the next command. Also, the frequency of user commands to tools may be much greater than the frequency of either user or tool requests to the Works Manager. In addition, the length of a Front End - tool/Foreman communication may be large. For example, in a typical session a user might request the use of a text editor (Front End - Works Manager communication), get a particular file to edit (tool/Foreman - Works Manager communication), and then insert two two hundred lines of program text into that file. Thus Front End - tool/Foreman communication is expected to vary form the infrequent, short request pattern to frequent, long transmissions of information.

- File Package - File Package

  Some very small fraction of these communications will consist of short, infrequent messages - e.g., a source File Package telling a destination File Package the length and encodement of a file - but the bulk of such communication will consist of files being transferred. Thus we can characterize this pattern as infrequent transmission of many bits.

The types of interprocess communication that were needed to provide the NSW functionality fell into three basic categories:

- Infrequent short transactions between previously unrelated processes (pattern 1).
- More frequent, continuing transactions between processes which maintain a relationship (pattern 2)
- Very long transactions or very frequent extended transactions (pattern 3).

## 4.1 MSG Design

MSG was designed to support these NSW patterns of communication by providing two different modes of process addressing:

- o generic addressing
- o specific addressing

and three different modes of communication

- o messages
- o direct communication paths (connections)
- o alarms

Generic addressing is used by processes which either have not communicated before or for which the details of any past communication is irrelevant. It is restricted to the message mode of communication. A valid generic address specifies a functional process class. When MSG accepts a generically addressed message it selects as a destination some process which is not only in the generic class addressed but has also declared its willingness to receive a generically addressed message. If there is no such process, MSG may create one. Pattern 1 communication is always initiated by the transmission of a generically addressed message between some pair of processes.

A valid specific address refers to exactly one process and this address remains valid for the life of that process.

Specific addressing may be used with all three communication modes. Specific addressing is used between processes which are familiar with each other. The familiarity is generally because the processes have communicated with each other before, either directly or through intermediary processes.

Message exchange is provided by MSG to support the requirements of pattern 1 communication and some pattern 2 communication. It is expected to be the most common mode of communication among NSW processes. To send a message, a process addresses it by specifying the address of the process to receive the message and then executes and MSG "send" primitive which requests MSG to deliver the message. When MSG delivers a message to a process it also delivers the name (i.e., specific address) of the process that send the message.

The second mode of MSG communication is direct access communication. A pair of processes can request that MSG establish a direct communication path between them. Direct communication paths are provided to support the requirements of pattern 3 communication, such as file transfers between hosts, and some pattern 2 communication, such as terminal-like communication between a Front End and tool/Foreman. The ARPANET realization for a direct communication path is a host-to-host connection or pair of connections.

The alarm mode of communication is supported by MSG to satisfy a communication requirement typically satisfied by

interrupts in other interprocess communication systems. Alarms provide a means for the process to alert another process to the occurrence of an exceptional or unusual event. Processes may send and receive alarms much as they send and receive messages. However, there are significant differences between alarms and messages. The rules that govern the flow and delivery of alarms are different from those that govern the flow and delivery of messages. In particular, the delivery of an alarm to a process is independent of any message flow to the process. That is, the delivery of an alarm to a process cannot be blocked by any messages queued for delivery to the process. Unlike a message which can carry a substantial amount of information, the information conveyed by an alarm is limited to a very short alarm code. This limitation implies that the delivery of alarms can be accomplished in a way that requires little in the way of communication or storage resources. This makes it possible for MSG to insure certain "priority" treatment for alarms which makes them suitable for alerting processes to exceptional events. While similar to traditional interrupts, alarms are different in one important respect: the delivery of an alarm to a process does not necessarily imply that the process is subjected to a forced transfer of control by MSG. For this reason, we have chosen to use the term alarm rather than interrupt.

NSW is implemented as a number of processes running concurrently on a number of different host computer systems. These hosts are heterogeneous systems with widely varying support

for the concept of "process" and the environment they provide for supporting processes. MSG on each host can be thought of as a uniform extension of the host's operating system. It provides the basis of the commonality necessary for the high level NSW components (e.g., Works Manager, Front End, etc.), which are developed out of the local operating system concept of process, to communicate with each other in structured and well-defined ways despite the heterogeneous environment. MSG is the abstract environment upon which the NSW system design is specified. As a consequence, every host participating in the NSW system must have an implementation of MSG to support its NSW processes. However, a host need not support all of the NSW functional components. Consequently a host's role in the system will be defined by the collection of NSW components it implements.

## 4.2 Process Addressing

It was decided during the design of MSG to support a global process naming convention. Under such as scheme, a process has a unique name which can be used for communication purposes by any other communicating process. This is in contrast to a relative naming scheme in which different processes use different names to address the same process. A major motivation for adopting a global naming strategy was to support the ability of processes to pass useable process names in interprocess messages without system intervention. This strategy has proven very convenient and valuable in easily developing and extending multi-process

transactions (i.e., more than two processes working on a single transaction) merely by including process names as data in function invocation messages.

NSW is expected to operate continuously, but individual hosts may not be continuously part of it. This can occur because a given host is not scheduled for continuous NSW service, or because the host has failed. We have defined a particular period of NSW service by a host as a host incarnation designated by: <host incarnation name>::=<host designator> <incarnation designator> where <host designator> uniquely identifies a particular host computer, and <incarnation designator> is an integer which indicates the particular period of NSW service by this host.

The host incarnation name is part of each MSG process name. The purpose of including the host incarnation in the process name is to provide a means for allowing the system (i.e., MSG) to easily determine whether a given process name refers to a process that may currently exist or to one that existed during a previous period of MSG service by the host computer in question. In particular, since global process names of communicating processes are sometimes stored for long periods of time before they are again needed to initiate a transaction with a specific process, system recognition of obsolete process names resulting from a host crash and restart relieves the components of this type of message validity checking.

An alternative to including a host incarnation field in the global process naming scheme would have been to never reuse a process name. This approach is now becoming popular by using large, monolithic, unique identification numbers. In essence, the use of a host incarnation field within a global process name is a means for providing decentralized generation of unique process id's, provided the incarnation field is sufficiently large to not require recycling. Although the field was fairly large, the specification did not require that the incarnation field be unique for all time. Selecting an appropriate scheme for "remembering" previously used incarnation numbers for a "sufficient" time was left to the discretion of the MSG implementers on the individual hosts. In practice, the issue of misaddressed messages due to system restarts has not been a problem.

A complete MSG process name is of the following form:

<process name>::= <host incarnation> <generic designator> <specific designator>

The generic designator is a string which characterizes a process in terms of its functional relationship to other processes, and is instrumental in selecting a process to receive a generic request. For example, processes with generic designator WM are candidates for messages which involve Works Manager functions. The specific designator is an integer which is utilized by the MSG implementation to ensure unique process

names. A process name is always unambiguous. At all times it either corresponds to a single process or is invalid.

It should be clear that the MSG concept of process name also includes notions of both physical and functional addressing. The host incarnation field serves as a means of locating the referenced process, while the generic designator indicates the role of the process in the higher level system structure. Including these concepts in the process addressing scheme makes it more difficult to "migrate" a process to another host, or to have processes play more than one functional role. Neither of these have been problems in the NSW experience, while the inclusion of these concepts in the programmer visible interface to process names has aided immeasurably in the understanding, debuggability, and traceability of failures relating to the component implementations.

### 4.3 Transaction Protocols

MSG provides support for process-to-process message exchange, and a means (generic addressing) for processes to initiate message communication with other previously unrelated processes. At the level of the MSG process interface, a message is an uninterpreted sequence of bytes.

Two additional conventions to provide the general rules for intercomponent interactions have been established to provide the general rules for intercomponent interactions. One convention,

NSWB8, specifies the data types and data encodement of the message contents. Since NSW is based on communication between heterogeneous host systems, these conventions are required in order to have data meaningfully communicated and universally understood.

NSWB8 defines seven data types and their representation within message communication. There are six basic data types for transmitting data (boolean, index (small positive integer), signed integer, bit string, character string and an empty data type), and an additional type (list) to provide extensible data structures. A list data structure contains a specified number of other data elements, including possible embedded lists. Higher level data structures can be communicated using an appropriate set of these primitive data types. For example, resource charges for a tool session are represented in NSW messages as a list of two items, the first item being an index representing the type of charge, while the second item of the list is an integer representing the dollar amount of the charge. All data communicated between NSW processes in MSG messages is typed according to NSWB8 conventions.

An additional set of conventions, known as the NSW Transaction Protocol or NSWTP, has also been developed to provide a uniform way of formatting messages used to initiate functions and to reply to function invocations. NSWTP is specified in terms of NSWB8 data items. The basic model which NSWTP supports

is that of a transaction whereby one process invokes a function or procedure in another process, and may at some later have time a reply to that request returned to it.

An NSWTP message is a list structure consisting of fields indicating the type of message (e.g. function invocation, reply to function invocation), a transaction identifier used to match requests with their responses, and parameters of the request or response including a well-defined placement of the name of the function to be invoked and its outcome (success or failure code and error message). In addition, closely associated with the NSWTP protocol are conventions that specify how to obtain additional information which may be required to complete a transaction but which was not or could not be specified when the transaction was initiated. Such information ranges from providing missing parameters, to disambiguating names, to confirming actions. Very often this information is supplied by an on-line user through his Front End process, but it can be supplied by any help process specified in the transaction. To flexibly support this form of "help" a process initiating a transaction may specify another process as a help process, to be called in certain well-defined circumstances to provide the additional information required to complete the transaction. If no help is available when required, the operation is aborted. In a typical use of the help mechanism, the Foreman might specify the Front End process as the help process in a file Lookup transaction with the Works Manager.

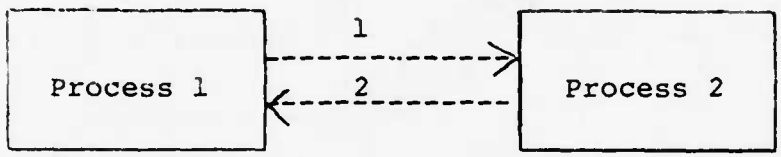
NSWTP is used to support interactions between NSW processes which do not require reply messages, in addition to the more widely used request/reply transaction paradigm. In addition, more complex patterns of component interactions, known as scenarios, are built out of combinations of request and request/reply transactions.

There are a number of frequently used patterns of transactions which are used to support the NSW functionality. Some of the simple and compound transactions typically utilized in the high level scenarios are illustrated in Figure 1.

### 4.3.1 NSW System Functionality

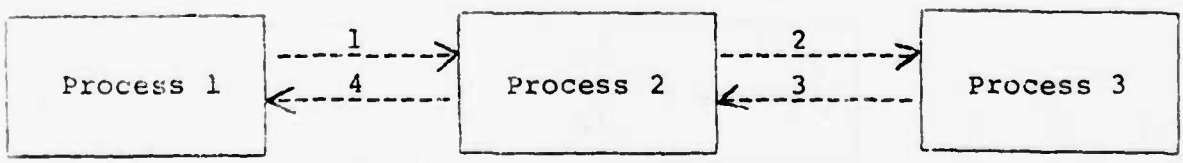
The previous section provided an extensive descriptin of the architectural framework upon which the NSW system is constructed. In this and the followign two sections we move up a level to decribe in some detail the functionality which NSW provides, and the roles of the various components in supporting that functionality. This section focusses on the system control software as embodied by the Works Manager components.

Figure 1A



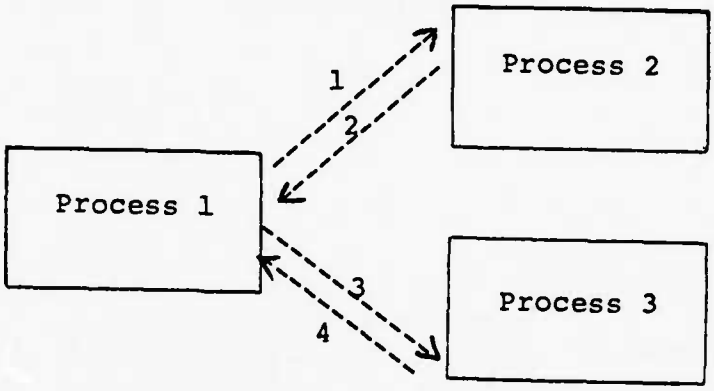

simple request/reply

Figure 1B



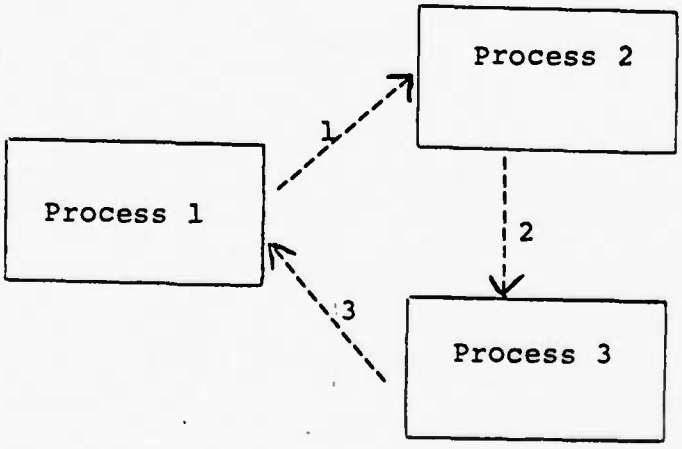

compound request reply

Figure 1C




sequential simple request/reply

Figure 1D




asynchronous compound request only

## 5. THE RESOURCE CATALOG MODEL FOR RETAINED OBJECTS

Much of NSW's concern is with the objects it manages, and the agents that attempt to access them. This section presents the basic notions relating to naming and manipulating NSW objects through the resource category. Because the naming and access control mechanisms used in the NSW Resource catalog have a number of unique features, we describe them in some detail. It was intended that the features for the catalog naming and access control mechanisms be useful for developing tools supporting project and configuration management.

Objects are the elements which make up NSW resource space. Examples of objects are files and services. NSW resource space is defined by the Resource Catalog. The Resource Catalog and the NSW software that uses it, in effect, implements a global symbolic name space for objects. NSW has been somewhat unique in its attempt at using a generalized associative retrieval data base facility as the basis for the implementation of the Resource . The object name lookup mechanisms have been tailored somewhat to this type of underlying catalog organization.

### 5.1 General Form of Object Name

The NSW resource catalog supports a single, uniform name space for naming all retained NSW objects. Objects are named as an ordered sequence of name components, separated by the special character "." It is often useful to view an object's name as a

path through the hierarchical NSW name space. A full NSW object name begins at the root of the hierarchy, and names a single terminal object through the sequence of ordered name components. In the NSW resource catalog, it is also true that each object is named by a unique complete name, i.e., it is reachable by exactly one path from the root.

The syntactic form of an object name in the catalog will be referred to below as an ObjectSpec (object specifier). An ObjectSpec always describes either one catalog object (if it exists) or no catalog object (if the object is not in the catalog). There is also a syntactic form that refers to groups of catalog objects, called a RegionSpec (region specifier). A region specifier uses a special "wild card" symbol (*) to designate the position in the name specifier which is variable and can match any object with zero or more intervening name components. ObjectSpecs and RegionSpecs may be rooted specifiers, in the sense that they contain the entire path from the root of naming hierarchy, or they may be unrooted and contain only a part of the path, the remainder to be supplied by the context in which they are used (i.e., by the scoping mechanism, to be described). The special symbol "$" prefixed to the first name is the covention used to indicate a rooted specifier

## 5.2 Keys

NSW access control is based on permissions held by the accessing agent. The representation of a permission within NSW is called a key.

Keys could conceivably be on a per object basis; that is, to access a file a user could be required to hold a permission for that file. There are two factors that make exclusive use of this approach infeasible for NSW:

1. File creation and deletion are expected to be very common operations. This means that the file name space will be a relatively rapidly and dynamically changing space. Consequently, rights on a per file basis would require a great deal of book keeping, both by users and by the system to keep rights up to date.

2. There are expected to be a very large number of files. Consequently, rights on a per file basis would require a very large number of permissions, even larger than the number of files when sharing is accounted for.

Accordingly, for controlling access to resource catalog objects, a key may refer to either a single object in NSW name space or all objects in a region of NSW name space, i.e., a key may be either an ObjectSpec or a RegionSpec. Keys are always rooted. Keys are stored within NSW in unevaluated form until an access test must be made, so the possessor of a key may use its permission to access objects created after the creation of the key. When a key is created, the set of objects in the catalog covered by the key (whether only a single object or many objects) may be empty. Keys are stored with the node record to which the permission applies.

There are different permissions for dispensing different kinds of access privilege. A single key grants the possessor only one kind of permission for the region of the catalog name space covered by the key. Three kinds of permissions are defined for reading and writing the catalog itself. These are:

- o LOOKUP permissions, required to do catalog object lookup operations in a given region (somewhat equivalent to directory read access on some systems),
- o ENTER permissions, required to create a new object name in a given region (similar to directory write on some systems),
- o DELETE permissions, required to remove a current catalog object name in a given region (a specialized form of directory write).

These permissions are applicable to all catalog operations regardless of the type of the object being manipulated. There are a number of other types of permissions which are applicable to one or more object types (e.g. execute permission, applicable to service objects).

In addition to these private keys, there is a system table recording public keys. Public keys are keys which system administrators have determined to be available for all users of the system. A user's permissions are defined by the union of his private keys with any public keys. Public keys are merely a simple mechanism to both avoid replication and support convenient update of keys common to all users. The regions covered are part of the "system".

## 5.3 Object Attributes

All objects in the NSW resource catalog have attributes including (but not limited to) "type" and "site", where "type" is the name of one of the system supported types (e.g., file) and "site" indicates the host on which the objects resides. Other attributes include an indication of the creating agent, time of last modification, file format, etc. Every NSW object has a unique name independent of its attributes. Objects cannot differ only in their attributes. The "site" attribute is used as a means for user control over selection and placement decisions.

## 5.4 Name Lookup

In general, manipulation of NSW objects proceeds in four phases:

1. name and attribute lookup
2. disambiguation (optional)
3. typed access control check
4. manipulation of the object itself

We can view the phases of object manipulation as a process of refining the set of objects which meet the specified requirements. Name and attribute lookup produces a set of one or more possible objects meeting the name and attribute constraints specified by a user. This set can be reduced to a single object via a search rule or user help via a disambiguation dialogue.

This single object is the result of the lookup procedure, which is then checked for appropriate access control based on the type of object found and the type of manipulation requested.

## 5.5 Naming Single Objects

All accesses to the NSW object catalog requiring an existing object use the same lookup procedures, which are as follows for the case where a rooted ObjectSpec (i.e., name beginning with "$") is given:

- Verify that the user holds a lookup key inclusive of the full name object specifier; if none, lookup fails.
- Lookup the named object and return its catalog entry handle; if no such object, lookup fails.
- If attributes are specified, succeed only if object found has the required attributes.

When referencing an NSW catalog object, a user need not always give a rooted ObjectSpec. Two mechanisms exist which allow some name components to be omitted when referencing an object. The mechanisms are scoping to support relative naming and ellipses to support omitted component names.

## 5.6 ELLIPSES

When specifying an object name (not a key or scope), an ellipsis represented by the special symbol "..." may be used to indicate Ø or more components may be missing from the name as specified, at the position of the ellipsis. Ellipses may occur

anywhere in an Object spec, but no two ellipses may be adjacent (they must be separated by at least one name component, e.g., $A...B...). Conceptually, an ellipsis signifies missing name components that the user expects to be uniquely defined by the portion of the object name that is explicitly given. If a specifier contains an ellipsis, and the system discovers upon checking the catalog that the portion of the name replacing the ellipsis is not unique, the user will be asked to select exactly one of the alternatives (if user help is available) or the lookup will fail (if user help is not available, e.g., name resolution during a batch run). For example, suppose the catalog contains only the names

1. $public.tools.TECO.RADC-20
2. $public.tools.TECO.ISIE
3. $public.documentation.TECO.RADC-20
4. $public.tools.Simula.RADC-20

The specifiers $...ISIE , $...Simula... , and $public.documentation.TECO... all refer to unique objects (2, 4, and 3, respectively). The specifier $...TECO... is ambiguous with three possible matches (1, 2, and 3) and the specifier $public...TECO.RADC-20 is ambiguous with two possible matches (1 and 3).

## 5.7 SCOPING

The resource catalog lookup function allows users (programs) to reference objects relative to user selectable regions of global NSW resource space. These regions which are used as part of the lookup function are called scopes and are referred to by ScopeSpecs (scope specifiers). Several ScopeSpecs can be combined according to parallel or sequential lookup rules (see below) to form a user's scope.

A ScopeSpec is a sequence of name components that designates a region of NSW name space, a context in which unrooted specifiers can be looked up or entered. A ScopeSpec is syntactically a rooted RegionSpec with exactly one instance of a "wild card" component (*). The wild card designator is used to match any number of unspecified name components in that position of the name specifier. Often, but not necessarily, the wild card symbol will occur at the end of the ScopeSpec. For example, $A.B.* and $A.*.B are valid ScopeSpecs, but $A.*.B.* and $A.B are not.

Under scoping, when a user provides an unrooted ObjectSpec or RegionSpec, it is substituted for the wild card symbol in a ScopeSpec to form a rooted ObjectSpec which is then subject to the standard NSW catalog lookup procedures. Naming an object relative to a ScopeSpec is much the same as directory-relative naming common on other systems. Each NSW user has a collection of ScopeSpecs, called the user's Scope, which supports all lookup

operations regardless of the NSW operation being performed. At any instant of time, a user's scope must contain at least one ScopeSpec, but may contain many more. The user's scope is maintained as a per-session data object, initialized from the user's permanent node record[2]. The scope can be modified either permanently in the node record or temporarily in the session record. It is not necessary for a user to have permission for objects in a scope when the scope is created or expanded, since permissions are checked only when the user attempts access to an object.

All unrooted ObjectSpecs and unrooted RegionSpecs use the scoping mechanism. The scoping mechanism is bypassed for rooted specifiers; prefixing the specifier with the special symbol "$" indicates that the name is rooted and should be looked up in the context of the entire catalog. The convention used throughout the NSW is that names which begin with a $ are relative to the root of the catalog, whereas names which do not have a leading $ are relative to the scope in effect at the time.

At any point in time a user's scope may be:

1. A single ScopeSpec
2. A set of ScopeSpecs in which ordering is unimportant. Current ScopeSpecs are searched exhaustively for all matching members.

---

[2] When a new node is created, the scope field in the node record is automatically set to a private "own" space which is associated with and created for the new node.

3. A sequence of ScopeSpecs in which ordering is important. Search ceases with the scopespec first providing a matching entry.

Case 2 is referred to as a parallel scope, whereas Case 3 is a serial scope.

A scope consisting of one ScopeSpec is analogous to a "working directory" on other systems. A serial scope is a slight generalization of search rule lookup, available on some systems. Parallel scoping seems to be unique to the NSW system.

## 5.8 Entering Catalog Object Names

A set of rules similar to those for lookup apply when entering objects into the catalog. As with lookup there are various modes for doing this: rooted and unrooted names, with and without ellipses.

Ellipses in the name of an object being entered in the catalog serve to indicate that the name is to be "completed" in the context of the current catalog using the incomplete name lookup mechanism described earlier. Unrooted names with ellipses are resolved within the context of the current scope only. The resulting rooted name with ellipses can only be used to replace an existing object. It can not be used to create a new object name. New object names must be fully specified. To add an object to the resource catalog an enter permission to the appropriate region is required.

## 5.9 Naming Groups of Objects

The name lookup and entering conventions discussed so far are intended to identify and name a single catalog entry. At times, there is also a need to succinctly name a group of catalog objects, usually in conjunction with some form of processing common to the entire group. The SHOW OBJECTS command is an example of where a designator is often used to denote a group of matching objects instead of being resolved to a single object. The syntactic form of a name referring (potentially) to several catalog objects is a RegionSpec. Keys and scopes are specialized RegionSpecs. The form of a RegionSpec is equivalent to that of a partial name specifier using ellipses, with the exception that the wild card designator represented by the special character "*" instead of "...", is used to denote reference to multiple objects. A general RegionSpec can have multiple "*" components, including preceding and trailing any specified name parts. The matching set can be reduced by including particular attribute values with the RegionSpec. For example:

```
$A.B.*.C/type=file
```

would refer to the collection of file objects to which the user had lookup access, which began with component names A.B and terminated with component name C. Parallel Scope rules applied to a RegionSpec generate a collection of objects which is the union of the RegionSpec applied to each scope region. Serial Scope rules applied to a RegionSpec generate the set of objects

matching the specification for the first ScopeSpec which has a non-empty lookup result.

## 6. FILE SYSTEM MECHANISMS

The NSW file system and file handling in general play critical roles in the functionality of the NSW system. Files are the dominant objects populating the resource catalogs and are a key item in the interoperability of NSW program services. This section discusses some of the principal elements of the file system mechanisms.

There are two distinct, but related storage systems for supporting the file storage requirements of NSW users and services. The NSW file system provides a long term, sharable, uniform space for files. Files in NSW space are managed by a combination of NSW core software and NSW host support software. Workspaces support service instantiation in NSW and represent a second mechanism for file storage which can be characterized as more temporary and private than NSW file space (although the system does permit workspace files to remain inactive indefinitely and to be remotely accessible). In addition, workspaces do not necessarily support the syntactic and semantic uniformity that NSW file space does. Attempts are made to provide a degree of uniformity in the treatment of workspace file management across service providing systems, but total uniformity across hosts is neither required nor expected.

The existence of two file storage systems in NSW is a direct outgrowth of implementation and efficiency considerations. In a heterogeneous host environment, it is impractical to

significantly modify either the existing operating systems or the available services, and it is undesirable from a performance point of view to support all of the distributed file system attributes for all file objects.

The heterogeneous systems which are the basis for NSW host software and the different levels of host implementation which the NSW design permits lead to a user model which exhibits some non-uniformities. For example, users are expected to understand that attempting to use services and hosts with different levels of commitment to the NSW system in an integrated fashion may require extra care.

## 6.1 The NSW File System

*Files stored in* the NSW file system have the following general syntactic and semantic properties:

1. All files are named with a common uniform NSW syntax regardless of the host on which the files were created or the host which currently stores the files.

2. File name lookup procedures for all file references employ all of the standard NSW file system conventions (i.e. scoping, unscoping, help disambiguation, etc. and are subject to standard NSW access control mechanisms. Each NSW file is sharable, lockable, subject to audit trail, possesses common types of attributes, etc. in a uniform fashion that is NSW-wide.

3. NSW software does not control access to any storage unit smaller than the NSW file. Nor does it implement standard access mechanisms for file i/o. Other than recording information which types the structure of a file object, and, where appropriate, performing translations of complete files based on this information, NSW is unaware of the internal

organization of file data and does not support programmable file access methods.

4. There is no a priori relationship between the name of an NSW file and the host that stores the file. Users are free to organize their NSW file space regions without regard to host boundaries.

5. Regions of NSW file space, into which NSW files are entered, are assigned to projects on a long term basis.

6. The NSW core software often automatically moves an image of an NSW file from one host to another to support remote access file references.

Because these features are implemented on a collection of large scale, heterogeneous host computers, accessing files stored in the NSW file system generally entails significant overhead.

## 6.2 File Images

Users can request that an information loss-less image of an NSW file be created and moved from its current NSW storage host to another NSW storage host (if such a transfer is possible) via the PLACE command. The system understands the equivalence of the original physical copy and the new physical image. Files entered (imported) into NSW file space are always marked as "original." Images maintained by the system as a result of a PLACE operation are marked as a "user-directed-image." In addition, to support the copy semantics for workspace file access, images of NSW files are often transported from a storage host currently supporting a physical copy to one which needs to use the file. When this occurs, and when the destination host is an NSW File Bearing Host, the destination may at its discretion retain an

information-lossless image of the original to serve as a cached copy. Images maintained by the system (for example, to support tool file access) are marked as "system-directed-images." The retention of a cached image is coordinated with the central catalog.

The NSW file system understands the equivalence of all physical file images, and can use any one to satisfy a user/tool file request. Users can direct the system to use a particular image by specifying the host attribute when referencing the file. Users alone are responsible for managing (i.e. moving, deleting) the disposition of originals and user-directed images (within allocation limitations, of course). Users are "charged" for storage associated with originals and user-directed images. The system manages the collection of system-directed images that it has decided to cache. A File Bearing Host may at any time, assuming the proper coordination with the central catalog, delete a cached image in accordance with its cache management policies and current file space demands. The central catalog process (Works Manager) may also initiate the deletion of cached images using the same mechanisms as for supporting user deletion of originals. Caching represents an area in which the system performance may be tuned. Users are never "charged" for system-directed-images.

On any lookup operation the user may limit the selection of an image through the host attribute. Without a specified host

attribute, the system selects any image interchangeably. If a particular image is specified but unavailable, the operation will fail. Replacing or otherwise modifying any image of a file catalog object invalidates all existing images of the file.

## 6.3 Workspace Files

NSW supports a copy model for workspace file processing. A copy is a snapshot of an NSW file at a given instant in time, allowing for the possibility of some host or service dependent data transformations from the original. The NSW system does not maintain the mutual consistency of workspace copies of NSW files. Optimizations support read only access whereby the file is not actually copied into a workspace when a locally accessible NSW file space image is already available. However, the semantics of file copying still prevail from the perspective of the accessing service.

NSW relies on a workspace copy model of file access as a means of achieving uniform access semantics for all Tool Bearing Hosts. In this way, the user/programmer view of a file reference is the same regardless of the availability of local NSW file storage resources on the host, and regardless of whether or not a information lossy, automatic tool specific file translation occurs. In addition, NSW does not provide host independent file access methods, placing further emphasis on integrating the NSW copy mechanism with local host file access methods. This

approach was motivated by the desire to make minimal modifications to existing software tools.

Files stored in NSW workspaces have the following properties:

1. Workspace files are individually and uniquely named using a syntax which includes host specific or service specific naming conventions.

2. Although some workspace files may be derived from NSW file system files (for example, were created by copying or otherwise processing NSW files), the NSW file system keeps no record of this relationship. Environments whereby a workspace file bears a definite logical relationship to an NSW file can be developed by users and/or tool interface software using NSW supported mechanisms.

3. All files in an NSW workspace are stored on the workspace host.

4. Since workspace files are not managed by NSW file system software, there are no mechanisms for supporting multiple images of them, or for automatically migrating images of them to other hosts.

5. Workspace files are private to those users who can access the workspace. There is no NSW access control for objects smaller than an entire workspace, although some workspaces may support host specific access control mechanisms for individual workspace files through the services that run in the workspace.

6. Workspaces assigned by NSW to support a user request are often only temporarily bound to the user. Hence, in many cases, workspace file storage will not represent the long term commitment for maintaining files that the NSW file system storage represents. NSW host interface software provides mechanisms for copying workspace files to permanently allocated storage areas prior to deallocation of temporarily assigned workspaces.

## 6.4 File Locks

Users/services can request that a "lock" be set on an NSW file to control access to it as it undergoes modification. Historically within NSW these file locks have been known as semaphores. A lock on an NSW file covers access to all images of the file. The duration of the lock can be indefinite (until explicitly cleared), or can coincide with the lifetime of the setting activity (service or user session). A permission to update the file object is required in order to successfully set a file lock. Locks can be set in conjunction with the request to access the file, or alternatively by using an independent lock request primitive. A lock request can be one of the following types:

- exclusive lock, whereby only the locking user can access the file object in any way
- exclusive write, warning on copy lock (EWWC), whereby only the locking user can modify the file object, but any user can obtain a copy of the file after an appropriate confirmation (provided of course that the actual file image is not being modified at the time)
- warning lock, whereby only requests which specify this form of lock will be accepted for any type of access

Exclusive locks are intended to support strictly private use of an NSW file object. EWWC locks designate a single modifying agent while allowing copies of the file to continue to be made after appropriate confirmation that this is desired. This type of lock is a direct result of the predominant copy mode of file access, leaving a consistent image maintained by NSW. A warning

lock is intended to support private user protocols for accessing files.

## 6.5 File Representation

NSW recognizes two aspects of file representation: the representation of the data elements within a file; and the structure of the file, that is, the way data elements are organized within the file.

Data representation types:

- Text (8 bit bytes, each byte from a standard text code; e.g., ASCII)
- Binary + data element byte size

File structure types:

- Sequential
- Record structure
  - Set of records - fixed size or variable size.
  - Data is sequential within a record.
  - Each record has a size and a name - the name may be numeric and implicit by the record's position in the file, or it may be a string.

The representational information for each NSW file is contained in the NSW catalog entry for the file, and is made available to the appropriate host support software component when access to the file is required. This information is the basis for file translations necessitated by the heterogeneous host environment.

## 6.6 File Movement from Host-to-Host

Files are moved from host-to-host in NSW for a variety of reasons. These include tools referencing an NSW file which does not have an image present on the local host NSW file storage area, and user commands initiating LISTING an NSW file. One of the basic models for file movement is as follows:

The NSW system component which initiated the file operation is returned a list of descriptors for all the available physical images of the file. Based on the location of the file images relative to the accessing host, one of the images is selected for use. The strategy used in NSW is to use a locally available image whenever available. If no local host image is available, an image on a host of similar type is the next preferable because of the availability of transfer modes which are optimized for the particular host type. This is known as a family copy. The transmission format for family copying is left entirely to the discretion of the software implementer for the host type. If no family copy is available then the file is retrieved from any available host using a standard NSW-wide transmission convention and encodement known as IL (interchange language).

To retrieve a file from a remote storage host, the accessing component initiates a transaction with a File Package component on the selected storage host. Using the physical file descriptor data obtained by the lookup operation, the accessing component communicates to the donor File Package the name and type of the

file to be transferred. The transfer takes place using appropriate conventions on a direct connection between the file donor process and the file receiver process. Foreman, Front Ends, and other File Packages are all at various times potential file recipients.

## 6.7 File Translations

File translation may be required to enable a file created by a service on one host to be used by a service on another type of host. The system supports and performs a standard set of file translations both between different data representations and between different file structural types.

the translations supported by the system include:

```
text -> text                    ;e.g., ASCII<->EBCDIC

text -> formatted text          ;e.g., ASCII ->
                                ;       any of a set of defined
                                ;       printer standards

sequential text ->              ;e.g., EOL -> EOR
        record structured
        text

record text ->                  ;e.g. EOR-> EOL
        sequential text

record binary ->                ;e.g., remove EOR's
        sequential binary
```

In NSW, file translations occur automatically. When a file is referenced the system decides what translation, if any, is required by means of heuristics that take into account the

structure and data types of the file, the nature of the destination host where the file will be used, and the nature of the service that will use the file.

## 7. PROVIDING PROGRAM SERVICES

One of the major features NSW provides users is the ability to access computational services on tool bearing hosts. Services are a type of object managed by the system.

The basic idea behind handling services is that a user is provided a uniform interface for invoking any service by its resource catalog name, or more precisely by a "service spec" which is looked up in the user's resource catalog context.

There are a number of different types of services the system supports. These include:

1. Single interactive programs. The user interacts directly with an individual program on a "service bearing host". The user is effectively logged into the host, running the indicated service in an available NSW workspace.

2. Service bearing host NSW command interpreters. The user interacts with an NSW style command language interpreter on the host, to manipulate NSW files and to run NSW services on the host. We call such a command interpreter a "workspace command interpreter" (WS-CI) because it operates in the context of a workspace on the host.

3. Service bearing host command interpreters. The user interacts with the standard native command language interpreter for the host, to manipulate files and run programs on that host within the context of an NSW workspace.

4. Service bearing host operating systems. The user interacts directly with the service bearing host operating system. This is known as ICP (Initial connect Protocol) access to the host, and is the equivalent of ARPANET access to a well-known socket address. The user is not automatically logged into the host, and there is no NSW workspace. A host need not have NSW software to be accessed in this mode. It need

only support ARPANET protocols. The most common well-known socket address is for ARPANET TELNET service, although others are defined and used.

5. Batch Services. The user makes use of NSW features to submit a job to a batch processing host. When a user invokes a batch service, an NSW component known as the Interactive Batch Submissin program (IBS) is actually runt to interact with him together the information required to submit the job (e.g. input files, etc.). Following that, the job is placed in the NSW batch job queue to be transmitted to the target host, executed, and the results returned to the user, all under NSW system control.

A user is able to invoke a service by the "use" command regardless of its type. For example, if the operating system for the RADC-Multics host was an ICP service cataloged in NSW name space as $PUBLIC.SERVICES.ROME.MULTICS

```
NSW: use rome.multics
...
```

would place the user in contact with the Multics operating system at Rome. Similarly, suppose the compiler for the BCPL programming language is cataloged as the program service $PUBLIC.SERVICES.BCPL. The command:

```
NSW:  use bcpl
...
```

would place the user in contact with a newly created instance of the compiler in a workspace.

For program services which are instantiated within NSW workspaces and which access files, there are two basically different modes of operation.

In native mode the workspace is used as a staging area for

copies of NSW files which the service requires during the service session. Through WS-CI commands, copies of the appropriate NSW files are moved to the workspace area. At this point, the service executes directly against the environment defined by the workspace(and the rest of the host) exactly as if it were executing outside of NSW. That is, the conventions in effect while the service executes are those of the workspace host, and not NSW. When the service completes, the user can copy any relevant workspace files back to NSW file space, again using a WS-CI. This is the simplest way in which existing host services can be provided access to files in NSW file space. It requires nothing more than software to support a WS-CI and the movement of files between a local workspace and NSW.

A second mode of operation is more integrated and requires NSW supporting software on the service host to mediate file references between workspace file storage and NSW file storage as the service executes. Services that operate in this way are said to operate in NSW mode. NSW mode services are characterized by their ability to manipulate both workspace and NSW files. Tool encapsulation is one of the techniques used in NSW to take programs originally developed to run as native to the host and convert them to operate fully integrated with the NSW system.

## 7.1 Tool Encapsulation

In general terms, NSW encapsulation implies the automatic trapping and translation of local host operating system calls into calls meaningful in the NSW system. Any trapping and translation is done within the Foreman component. Using an encapsulation technique, we take programs which are written exclusively for the local host operating system execution environment, and with little or no modification execute them as NSW tools. This is possible only because of the similarity, in many aspects, of the NSW system to a conventional single host operating system. As an example, when an encapsulated tool issues a local system primitive to gain access to a file, the Foreman gets control and translates the request into one which provides access to an NSW file. This assumes that the "old style tool" is somehow capable of handling the NSW filename syntax within the local host file manipulation primitives. In many operating systems, TOPS-20 for example, this is often very easy since the tool will frequently allow the "system" to gather the filename from the user.

Under encapsulation, the Foreman is interposed between the tool and the operating system for selected operating system functions. With its intimate knowledge of both the local system primitives and the NSW system structure, the Foreman provides the NSW program execution environment using both local host facilities and facilities supported by other NSW components.

Encapsulation cannot be discussed in terms of its algorithms. It requires an extensive knowledge of the local host operating system primitive operations, and a determination of how they can be made to relate automatically to the NSW environment. Thus each Tool Bearing Host approach to encapsulation is somewhat different. As far as the other NSW components are concerned, running an encapsulated tool is no different from running any other type of tool. Generally the tool initialization and termination conditions, interactions with the file system, and the communication with the tool user will all require careful attention within the encapsulation component of the Foreman.

A more limited form of encapsulation (one that employs only automatic pretool execution Foreman processing and automatic post tool termination Foreman processing to reintegrate the tool session results with the NSW system) has also been used successfully by some Tool Bearing Host implementers. While this mode of interfacing may not be as integrated with NSW conventions as a more complete encapsulation and may not be appropriate for highly interactive tools, it does allow existing software to be inserted without extensive modification.

## 7.2 Conversational Partners

At various points during an NSW session, a user may converse with either an NSW command interpreter, a workspace command interpreter or a service. These three conversational partners

perform similar functions, namely accepting user requests for some action, and then carrying out the requested activity within the context which the partner represents. They are distinguishable in the commands they support, in the domain over which the commands take effect, and in the form and style of their interaction. To a rough approximation, the three levels of conversational partner represent NSW's hierarchical refinement of network-wide, host-specific, and service-specific contexts.

Within the NSW command interpreter context, the user can expect to see a strict and uniform adherence to NSW conventions for all commands and interactions, with the exception of commands provided as interfaces to other contexts (e.g., the command to import files into NSW file space from a host file system). All commands are executed within the NSW session context. There are a set of uniform commands for entering ("use" and "resume") and returning from (^N) the more localized workspace or specific service contexts.

A workspace command interpreter (WS-CI) context has aspects which are uniform across all WS-CI's, and others which are tailored to the host on which runs. There are a number of similarities between some of the commands supported by the NSW command interpreter and some of the standard WS-CI commands, for example the commands for starting a service or copying a file. The essential difference is the context within which the commands are interpreted. Whereas NSW level commands are interpreted in

the context of the entire NSW object catalog, WS-CI level commands are normally processed in a context limited to the workspace and the workspace host. Additionally, WS-CI's support commands which are in part oriented toward NSW objects and in part oriented toward workspace objects in order to provide a standard interface between the two file spaces. WS-CI's may also be individually extended in ways which reflect the nature of the facilities available on the support host. There is a standard way to enter ("run") and return form (^C) a service environment via a WS-CI.

Command interpreters which are part of services are the least standardized among the potential NSW conversational partners. Since the NSW command interpreter and the WS-CI are both part of the NSW system software, uniformity can be required and achieved. Since service software is, for the most part, developed outside of the NSW context there is less control over the conventions used. Conventions for different services can be expected to vary, even when the services run on the same host computer. Services also may vary quite a bit in their size and complexity, ranging from a service with a single command to services which include their own internal data and file management mechanisms.

# 8. USER INTERFACE SOFTWARE

The Front End is the user's interface to the NSW. An NSW Front End performs several distinct functions.

## 8.1 Command Interpretation

The Front End supports the NSW command language, the means by which a user interacts with NSW. Command interpretation involves parsing user typein and initiating the NSW system operations required to satisfy valid commands. To date there have been two implementations of the Front End - one that runs under the TENEX and TOPS-20 operating system, and one that runs under UNIX. The NSW command language is uniform across all implementations of it.

## 8.2 Interaction with NSW System Components

In order to satisfy user requests the Front End interacts with other NSW system components. These interactions are governed by the set of NSW protocol "scenarios". In addition to the scenarios which support the major functional aspects of NSW (e.g. running services, manipulating files), there are a number of scenarios concerned with supporting aspects of the user interface to the NSW system. These scenarios are to a large degree associated with obtaining the status of various NSW components and data bases which support the system implementation.

The Front End is designed to operate in two "command return" modes: "deferred return" mode for which the Front End retains control until the command is completed; and "immediate return" mode whereby control is returned to the user immediately after a protocol scenario for the command is initiated rather than after the scenario completes.

The immediate return mode permits the user to initiate other commands while the protocol scenario for a command is being performed. When the scenario completes the user is notified that the command has completed. The user displays any output produced as the result of the command execution when, and if, he wishes by an explicit or implicit "display" command. Immediate return mode was implemented in direct response to the increased delays associated with NSW operations requiring substantial interhost activity.

Usually the communication path between a user's Front End and a service is a TELNET connection, although other forms of communication have also been supported. A user may have multiple tool sessions active at the same time. To support multiple tools, the Front End provides a means by which a user may switch his attention back and forth among the various active tools and between the tools and the NSW command interpreter, while maintaining the proper context for those activations currently unattended.

## 8.3 Terminal Control

The user and the Front End interact by means of the user's terminal. The Front End exerts control over a number of basic terminal handling functions, such as the manner in which various non-printing "control characters" are represented when echoed and output, the input characters that may cause "program interrupts", and so forth. In addition, the Front End provides the means by which a user may uniformly edit his typein before the Front End acts on it.

## 8.4 Command Procedures

The command procedure feature permits a user to define "composite" commands which consist of a sequence of NSW Front End commands which are to be executed as a single command.

The definition of a composite command or command procedure is a text file stored in the file system. When a command procedure is invoked by the user, the corresponding definition file is retrieved from the file system and interpreted by the Front End. Command procedures are able to gather input (e.g. responses to NSW "help" messages) from the user's terminal as part of their execution. Command procedures were introduced in part to cope with the repetiveness of a number of NSW operations and in part to reduce the requirement for human interaction during times of extensive network delay.

A particularly important command procedure is contained in the Login command file. This file is the repository for a set of user specified commands which are automatically executed by the Front End whenever the user logs into NSW. The Login command procedure enables a user to conveniently store information about his preferred use of NSW and to have the Front End set various system usage parameters for him automatically when he logs in.

## 8.5 Access to NSW

To access NSW a user first obtains an Front End and then logs in. One way Front End's may be accessed by users is through an ARPANET ICP exchange. In this sense, obtaining an Front End is similar to accessing an ICP service. The user must instruct a program acting on his behalf to engage in an ICP exchange with a host that provides NSW Front End service. This may be done explicitly by specifying a host and ICP contact socket, or implicitly by invoking an "NSW contact" program that knows how to obtain an Front End. A Front End obtained in this way is known as a "dispatched" Front End.

After obtaining an Front End process, the user interacts with it through the ARPANET TELNET connection established by the ICP exchange. The Front End interacts with other NSW modules by means of MSG communication. When using a program service the user interacts with it through two TELNET connections, one connection between the terminal and the Front End and the other between the Front End and the service.

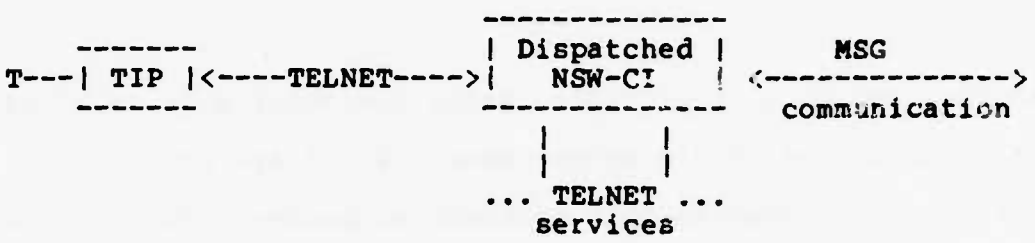



FIGURE 2.

This configuration is shown schematically in Figure2. In the figure the access host is shown as an ARPANET TIP, but it could be any other host that supports user TELNET.

One of the problems with this configuration is that the user is two TELNET connections removed from any services he uses. This is the case even when the service used is on the host used to access NSW. This is clearly undesirable for reasons of efficiency. Perhaps more importantly, with this configuration the ability to use local high speed communication paths and protocols, such as employed by the IBM 3278 full screen mode of interaction, is lost. On the positive side, the dispatched Front End can be supported on one of the otherwise important NSW hosts (e.g. Works Manager host). This minimizes communication costs between the Front End and (say) Works Manager, and also minimizes the number of different hosts needed for special purposes like debugging and testing. The NSW system supports an Front End on

TOPS-20 host which is dispatched from a well-known ARPANET ICP contact socket.

Another approach to providing Front End service is to place the Front End on the user's access host. There are several difficulties with adopting this approach exclusively for all user access.

1. The Front End interacts with other components by means of MSG. Each access host would need an implementation of MSG. This is a substantial undertaking, and which would be impossible for some access hosts, such as TIPs.

2. The command interpreter function would need to be re-implemented for each access host. Apart from the expense, it would be difficult to achieve uniform implementations.

However, having the Front End run in the user's access host is a viable approach if the access host is one that is relatively inexpensive so that there can be many installations of it, or if there are many installations of it because the host is very useful for other purposes. Recognizing this, the NSW project has designed and implemented an enhanced user-access machine using a PDP-11 UNIX host as a base. This configuration is known as a UNIX NSW Front End. The configuration for this alternative is shown in Figure 3.

The UNIX Front End provides the same basic functionality as provided by a dispatched Front End on a TOPS-20 host. However, since UNIX is itself a powerful host operating system which supports a variety of sophisticated services, it is reasonable to

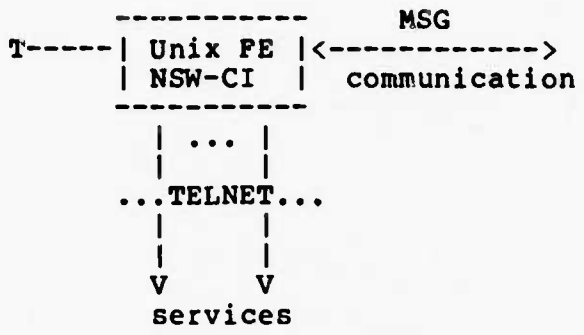



FIGURE 3.

allow users whose access point to NSW is a UNIX Front End to access these UNIX services through their UNIX Front End. Such services run under the control of the UNIX Front End in a fashion similar to accessing other NSW program services. To effectively support services on UNIX Front End host required a means for moving files between UNIX and the NSW. Accordingly the UNIX Front End has been enhanced with functions permitting limited integration with the NSW file system.

## 9. RELIABILITY CONSIDERATIONS

As noted in Section 2, early versions of NSW were extremely fragile and sensitive to many unanticipated events in the complex network environment. The loss of user files due to the fragility of the system was a common occurrence. To alleviate this problem an effort was undertaken to enable the system to robustly handle situations arising from host, network or component crashes or malfunctions. This was accomplished by adding some new component interaction "scenarios" and augmenting other relevant existing scenarios of operation toward improving system reliability. Together, these system enhancements became known as the interim reliability scenarios. They were referred to as "interim" because they were intended to be replaced by a more pervasive approach to overall system reliability. The so called "full" reliability plan was never implemented. The interim plan was implemented and enhanced with succeeding versions of the system. It resulted in a significant improvement in system reliability and represented an interesting (at the time it was implemented) approach to providing for reliable operation in a distributed system. complex situation. Some of the key aspects of the system reliability enhancements which are operational in the current NSW system are described below.

The goal of the interim reliability plan was intentionally limited. It was an attempt to guarantee that system malfunctions (other than catastrophic disk failures) will cause few, if any,

user files to be lost. This guarantee extended both to files stored in the NSW file system as well as workspace files which were "closed" but undelivered or temporarily undeliverable due to the malfunction.

Recall that the basic model of an NSW computation has a tool obtaining workspace copies of appropriate NSW files, modifying them as necessary by the particular application, and then delivering the modified files back into the NSW catalog when the tool completes. Two types of failures were addressed by the NSW interim reliability plan: failures that prevented the workspace files from being delivered to the NSW resource catalog, and failures that caused the catalog to "lose" a file after it was already "successfully" delivered.

There are a variety of situations which could cause the first type of failure. We briefly mention a few of these.

- Failure of the NSW host software or Tool Bearing Host system before initiating the file delivery operation
- Host inaccessibility of the Works Manager host preventing acceptance of the file at the NSW resource catalog
- A failure at the user access point preventing completion of the tool session
- A network failure preventing communication between appropriate NSW system components for completing the tool session.

The second type of failure occurs primarily as a result of a resource catalog host failure before steps have been taken to

assure that the new catalog entry has been written to storage that can survive the host crash and subsequent restart. Both types of failures are addressed by a single, integrated reliability plan.

## 9.1 Data Base Checkpointing

In order to guarantee that NSW file system files not be lost (except under rare circumstances) it is necessary to preserve the NSW file catalog.

Since the catalog is continuously referenced and updated, most of the time part of it is in volatile storage (i.e., main memory). To ensure that new or newly modified resource catalog entries are copied to non-volatile storage relatively soon after they are entered into the catalog a lock is taken on the entire catalog periodically (20 minute intervals). The lock prevents any Works Manager from initiating an interaction with the catalog, but it allows any ongoing interactions to proceed to completion. When all such ongoing interactions have completed, the entire catalog is copied onto non-volatile disk storage. The lock is then released, and new interactions can be started by Works Manager processes.

This mechanism is sufficiently inexpensive that its invocation at 20 minute intervals does not add significantly do delays. The 20 minute interval does, however, introduce a window during which the results of a successfully completed file

transaction may be lost. That is, since parts of the catalog are in volatile style, should the Works Manager host crash, any change to the file catalog since the last checkpoint may be lost.

In conjunction with this checkpoint, Tool Bearing Host and Works Manager software were extended to ensure that resources allocated to a tool session (e.g., workspace files) are not discarded until any catalog modifications resulting from the tool session have been securely saved by a checkpoint.

## 9.2 Saving Workspaces

Tool Bearing Host Foremen are responsible for maintaining workspace file data in non-volatile Tool Bearing Host memory. This data would normally be deallocated on tool termination. However, under the NSW reliability plan, this data serves as the redundant backup data which may be needed to prevent loss of user files should the Works Manager host crash after files have been delivered but before the next catalog checkpoint. To allow for the eventual deallocation of the workspace file data, a "guarantee" message is added to each tool termination scenario indicating that a checkpoint which includes the results of the tool session has been completed. When receiving such a message from the Works Manager, a Tool Bearing Host Foreman can assume that delivered files are "safe" from a system crash, and may deallocate all of the workspace resources. Because of the interval between catalog checkpoints, the guarantee phase of the

tool termination sequence can occur as much as 20 minutes after tool termination has been initiated. In the absence of such a "guarantee" message, the Tool Bearing Host Foreman is responsible for maintaining the workspace contents until it can successfully deliver the tool session results to the catalog.

Failure to receive a guarantee message from the Works Manager is only one of a number of events that may cause the Tool Bearing Host Foreman to initiate actions to save the workspace contents. Any NSW failure during the tool session will trigger similar action by the Foreman. An NSW tool session which did not successfully complete (from the NSW system perspective not the tool perspective) will be saved, and Tool Bearing Host software will periodically attempt to report it to the Works Manager until the saved status is acknowledged. The Works Manager in turn will report any newly saved tool sessions to the appropriate user, immediately if he has an active user session, or else on his next log in. Once a saved tool session has been recorded by the Works Manager and indicated to the invoking user throrough his Front End, the user has several options. Through appropriate Front End commands, he can either continue the execution of the tool within the workspace (assuming the tool has an appropriate restart entry point), or cause immediate delivery of any saved workspace files, or have the saved tool session deallocated with no further action.

## 9.3 Tool Bearing Host Restart and Data base Resynchronization

There are a number of distinct patterns of failure recovery to support the reliability model outlined above. Because of the centralized control structure represented by the Works Manager, an outage of the Works Manager host can block the completion of many on-going service and file operations, and also block the initiation of new requests requiring global NSW resources. To ensure the immediate re-integration of NSW host resources with the centralized system control functions after the Works Manager host is restarted, the Works Manager broadcasts an "I am now functioning" message to the Tool Bearing Host systems. This prompts the Tool Bearing Host systems to synchronize their current tool session data bases with a similar data base maintained by the Works Manager so that tool sessions that were successfully saved while the Works Manager was unavailable can be reused as indicated above.

When a Tool Bearing Host itself becomes unavailable, due to operating system crash or scheduled shutdown, all active service sessions are necessarily prematurely terminated. In a similar to a Works Manager host restart, as part of a Tool Bearing Host restart NSW software on the host contacts a Works Manager in order to reintegrate the host into the operational NSW system. This involves data base synchronization and reporting tool sessions which remain accessible to the appropriate users.

## 9.4 Failure Detection

One of the difficult problems encountered in developing a system like NSW and distributed systems in general is that of detecting component or system failures. In some cases there are signals passed from one layer of the system to another indicating the failure (e.g., a "carrier off" signal indicating a break in communication). However, in most cases there is either no such signal, or the signal is unpredictable in its delivery characteristics and warrants a complimentary mechanism. This is the general problem of the incompleteness of protocols which rely on negative acknowledgements and for their correctness.

The mechanism most widely used in NSW to detect component failures is the use of postive acknowledgements and timeouts. Basically, whenever a component initiates a request for which it expects a response, it sets a timeout interval which is the maximum waiting time before declaring that the communicating component is not operational and recovery (such as the reliability scenarios previously noted) should be initiated. In a system like NSW as implemented on the ARPANET and its constituent hosts, establishing useful intervals for timeouts is itself a technical problem. The extreme variability in the communication delays, in the loading of the participating hosts, in the number of NSW components required to participate in a scenario, and in parameters of an operation (e.g. size of a file to be transferred), all contribute to the unpredictability in response times.

Because of these problems, two mechanisms have been developed to enhance the use of timeouts for detecting failures in NSW transactions. These mechanisms are complimentary to each other, addressing the problem from the perspective of both the initiating and the responding process. The mechanisms are the intermediate status reply and the status probe.

## 9.5 Intermediate Status Replies

The concept of a partial reply for long transactions was introduced to avoid the timeout of a transaction by the initiating process or the intermediary processes of a compound transaction due to an unexpectedly long event, such as the transfer of a large file under heavy load. Transactions are designated as "short" or "long". Long transactions are those whose completion time is likely to have a large variance beyond the normal variance of message transmission and simple processing. For long transactions, the replies are broken into two phases: the first phase reply is an intermediate status response indicating only that all of the resources (hosts, processes, files, etc.) needed to complete the transaction are both currently available and committed to this transaction; the second phase reply signals the normal completion of the transaction. The responsibility for initiating the first phase status reply lies with the last process invoked in any compound transaction chain. Status replies are sent to the process designated in the invoking message as the one to which final

replies are to be sent. This is the process which typically times out the final response, and is usually the invoking process.

It is the responsibility of all intermediary processes in a designated long transaction to relay (and enhance) the status reply to any process which may be awaiting it. In general, one or more intermediate status replies are acceptable (but not required) for any transaction using NSWTP conventions. An NSW component will commit a longer timeout interval to those transactions which have indicated progress through one or more intermediate status reply messages. Figure 4 is a model for the introduction of intermediate status replies into a protocol scenario.

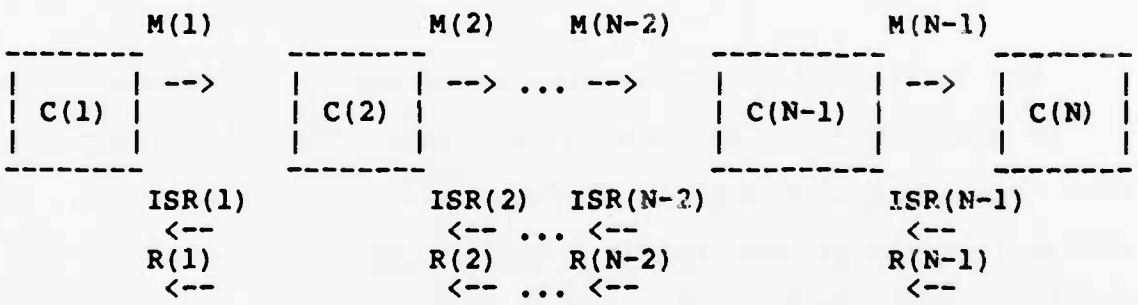



FIGURE 4.

The diagram above represents a chain of N components. Component C(1) initiates a protocol scenario the, completion of which requires N-1 additional components to be activated. Each additional component C(i+1) is activated by sending a message M(i) to it. Each message M(i) has as an argument the transaction

id tid(i) generated by component C(i). The ISR's are shown returning to C(1) from C(N). When the operation is complete, a chain of reply messages R(i) is sent from C(N) to C(1).

By allowing the servicing process to initially and periodically substitute status replies in place of final replies we provide some tangible evidence to the invoking process that the appropriate NSW components are still functioning. However, to make the procedures for recognizing component failures independent of any chosen frequency of status replies from the servicing process, an invoking processes (i.e. the process maintaining the timeout) has the capability to initiate, at its discretion, an operation which provides positive acknowledgement that a component failure has not occurred. These operations are called status probes.

One difficulty in introducing these status probe operations is the use of generic addressing for initiating NSW transactions. When this addressing mode is used the exact name (specific address) of the process handling the transaction is unknown to the invoking process at the time the transaction is initiated. Thus, requiring an intermediary status reply which cascades over all of the processes of a long transaction actually accomplishes two things:

1. There is some relatively immediate feedback that the operation can be initiated and will proceed subject to loading factors on the hosts and the network.
2. The initiating processes are provided with the specific

addresses of their generically addressed counterparts. This specific address can then be used to initiate further status queries directly to the appropriate servicing process.

## 9.6 Handling Timeouts

Associated with the two phases of response for long transactions are two timeout intervals. The short timeout interval is intended to protect against waiting too long (tying up resources) before deciding that the operation is not likely to complete because of resource unavailability. The long timeout interval is a low overhead approach to protect against failure during an operation, while allowing adequate time for the operation to complete under variable load and size factors. A component would commit to the larger timeout because it has the prior knowledge from the intermediate status reply that the operation has been successfully initiated, as well as the knowledge of the relevant operational parameters (e.g. size of file).

Whenever a long timeout expires, NSW components are required to initiate a status probe of the appropriate servicing process and to renew the long timeout should the target process promptly respond with a positive status reply. This response will indicate that the component is still working on the request and that aborting the transaction may be premature. NSW user interface software is also programmed to allow users to initiate status probe operations at their discretion for additional

feedback that the operation is progressing. In general, components with direct contact with a user can set very liberal timeouts provided the user is not locked out during the waiting period. Components in direct contact with users also employ a user to guide the software in its decision to abandon a transaction due to timeout.

## 10. PERFORMANCE CONSIDERATIONS

The NSW system was a complex undertaking, addressing a number of new and difficult system issues simultaneously. Because it was a prototype system, every effort was made to make extensive use of "application" level code to serve as the system implementation. A ground rule of the implementation was that modifications to the operating system kernels of the constituent host were to be kept to an absolute minimum. In the early design and implementation stages there was little consideration of or concern for the overall performance of the system. All of the initial attention was paid to developing the new distributed system technology. After a short while, it was apparent that the performance of the system did not compare favorably with that of current timesharing hosts. A significant part of the last few years of the NSW project has been devoted to understanding and improving the performance of the system.

The complex nature of the NSW system implementation made the evaluation of the performance of the system a difficult task. For example, even when the NSW is mapped onto a single host it involves a number of time-sharing jobs, each with multiple, parallel processes running on a large and complex operating system (TOPS-20). Systems like NSW had not been built before and very little support software was available for measuring its performance. The few tools that were available only measured selected parts of the performance characteristics of isolated

parts of the system. Accordingly, the first major thrust of the performance evaluation phase was the instrumentation of both the operating systems and NSW system components to define and capture relevant system performance parameters. After the effort to instrument the system and understand its performance properties, the project began a multi-year, multi-faceted approach toward improving the performance characteristics of the NSW system. This section briefly outlines the measurement and performance improvement phases of the NSW project.

## 10.1 The Measurement Task

To understand the nature of the performance measurement task and the tools used in that task, one must first understand some of the implementation details of the NSW system.

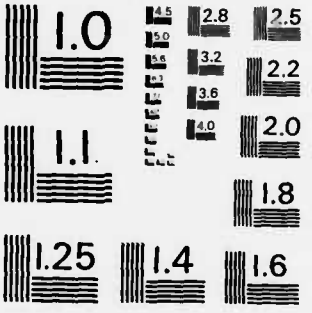
As noted previously, the NSW system is functionally decomposed into units known as the Works Manager, the Front End, the Foreman, and the File Package. These components interact with each other via the MSG interprocess communication facility in well-defined patterns governed by the NSW system protocols. In this manner, the location of the cooperating components relative to each other is completely transparent. Interactions among the components are grouped together into patterns called "scenarios". A scenario consists of the NSW process interactions required to implement a system operation, such as starting an NSW tool or copying an NSW file. Each NSW component is implemented

as ordinary application code, and as far as the host operating system is concerned, it is an individual entity making resource demands on the system. It is the programmed cooperation among these otherwise independent elements that implements the NSW system. The host operating systems that run these elements are unaware of the NSW.

There are two classes of tools which were developed and used for NSW system instrumentation. The first class of tools views the collection of concurrent activities on a host from the perspective of the host operating system. They serve to focus attention on the overall demand placed on the NSW host operating system by the individual and collective NSW components for the host. The second class of tools instruments the system from within the various components themselves and is organized around the performance of the higher level abstractions developed by the NSW software in carying out the NSW workload. They serve to focus attention on the NSW system performance from the perspective of the NSW user.

The NSW Works Manager is implemented only for the TOPS-20 (TENEX at the time of the instrumentation effort) family of computer systems. There is a Foreman, File Package, and MSG implementation for each Tool Bearing Host, including TOPS-20, MULTICS, and IBM-VMS systems. There are Front Ends for both the TOPS-20 and PDP-11 UNIX systems. Thus, a functionally complete "NSW system" can be configured using exclusively TOPS-20 NSW

AD-A132 320 A TECHNICAL OVERVIEW OF THE NATIONAL SOFTWARE WORKS(U) 2/2
BOLT BERANEK AND NEWMAN INC CAMBRIDGE MA
R E SCHANTZ ET AL. MAR 83 BBN-5238 RADC-TR-83-80
UNCLASSIFIED F30602-81-C-0213 F/G 9/2 NL




MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS - 1963 - A

components running on a single, self-contained host system of this type. For a variety of reasons, most of our initial measurement activity focused on such a configuration.

The following were some of the tools used in the measurement of NSW system performance:

- o JSYS Measurements

  JSYS measurement was a tool to record the number of system call (JSYS) invocations and accumulate the processing time in system context on a per-system call basis cumulatively for all processes of a specific job. Data is accumulated continuously for the given job from the time the facility is enabled until it is explicitly disabled. Using this facility, we were able to focus our attention on those parts of the application/operating system interface which were used most frequently and/or consuming the most resources.

- o Process State Sampling

  The process state sampling mechanism was used for the purpose of characterizing the various components of delay within the operating system which contribute to the overall response time. In a modern operating system, there are a wide variety of factors which contribute to the delay in execution of any given task. Some of these are dependent on the specific task while others are a reflection of the multiplexing of system resources among competing processes. The process state sampling broke down the various forms of delay for each NSW component, indicating the percentage of time spent in the various wait states.

- o System and Job Performance Data

  This program recorded periodic samples of data collected by the operating system during its normal operation. Among the pertinent values reported are the monitor's estimate of each process' working set size at the time of the sample, the scheduler.queue to which the process currently belongs, the fraction of the sample interval spent in various system overhead routines, the average page traffic to the drum and disk, and the fraction of the processor sold to each accounting group.

- MSG Event Logging

  The MSG communication subsystem was instrumented with an event logging capability which time stamped and recorded performance data at each major event for all of the MSG processes on a given host. Major events included the initiation and completion of communication operations from subordinate processes (e.g., sending and receiving messages) as well as the allocation and deallocation of processes to support generic classes. Additionally, for each event, the CPU time utilized by MSG and by the inferior component, and samples of the current paging activity counters for both were recorded. With the knowledge of the NSW component protocols, measures of NSW scenario performance for each component, as seen from MSG, were computed.

- The Performance Monitoring Package

  The Performance Monitoring Packages (PMP) provided application level components with a tailorable mechanism by which certain resource utilization characteristics of the programs could be measured and automatically processed. PMP consists of two basic programs: one which implements a form of event logging, and the other which processes the event logs and prepares analyses according to flexible format definitions. PMP packages were used to define and measure processing intervals relevant to the overall NSW system function or to the appropriate abstractions in a particular component implementation.

- Page Access Monitor

  This tool was developed to record the virtual memory page reference strings for the various NSW system components. A tool of this sort was needed to help understand the interplay between the functions being computed by the components and the demand paging memory multiplexing strategy used within the operating system.

- NSW Script Driver

  This tool was developed to automate the introduction of system workload. It simulated and coordinated the use of the system by multiple NSW users. The tasks which were to be executed were taken from scripts contained in files prepared off-line from the experimentation. The scripts were carefully designed to exercise particular parts of the NSW functionality.

## 10.2 Evaluation and Analysis

Using these tools, a number of experiments were undertaken to measure and evaluate the NSW system, its system components, and the hosts supporting those components. The results of the experimentation were reported in depth in a number of technical reports. Here we highlight a few of the conclusions from the performance evaluation phase.

The NSW performance experiments showed that there was no single system variable which could be used to accurately reflect NSW performance or which could be identified as the predominant system bottleneck. Rather, it appeared that the explanation of NSW performance characteristics lay in the extensive consumption of many host system resources by the cumulative NSW components in carrying out the complex series of interactions involved in the various NSW operations. This was further compounded by a host operating system environment which, partly because it is optimized toward multiplexing a series of unrelated processing requests, could not adequately handle a transaction oriented subsystem with demands comparable to those of NSW.

The experiments also confirmed our expectations that one could achieve excellent NSW response characteristics with a large enough single host configuration and a limited load. However, the amount of the computer resources required by NSW processes to carry out NSW operations, the causes of which were many, severely limited the achievable throughput and made such configurations somewhat non-cost-effective.

Only large host NSW configurations were able to adequately handle the large component processing demand, and then only for relatively few users (compared to "normal" usage patterns for the host configuration) before queueing delays drove response times quite high.

## 10.3 A Closer Look at Resource Demand

This section takes a closer look at one of the major resource demands (processing time) for a typical NSW transaction in order to understand the components of the heavy demand. The NSW GET FILE operation initiated by the Foreman process, which is similar to "open file" operation in a conventional host, is used.

The following table breaks down the per-component CPU demand for the participants in a single *GET FILE* scenario for a typical NSW host system. This CPU time is for an unambiguous file reference requiring no inter-host file movement.

| component | net CPU | |
|---|---|---|
| Foreman | 68ms | |
| MSG-FM | 100ms | (2 messages 50ms/message) |
| Works Manager | 494ms | |
| MSG-WM | 200ms | (4 messages) |
| File Package | 453ms | |
| MSG-FP | 100ms | (2 messages) |
| Scheduler | 100ms | (1) |
| | 1515ms | |

Table 1: Individual process CPU demand for a single NSW GET FILE Transaction.

Twenty six percent of the total CPU time is spent transferring messages between components. This message passing is analogous to more familiar subroutine linkage or domain crossing in most programming languages. However for the current NSW implementation, the "linkage" costs are actually even higher than the cost of plain message passing. For example, the messages themselves follow NSWTP, the NSW-wide standard encoding scheme which is foreign to the internal format of all of the host system. Thus, the form of messages and data passed in them require encoding and decoding by both sender and receiver for every message. Also added to the linkage cost is the approximate 100ms spent in scheduling the CPU for the logically integrated processes. Because of the transactional nature of the NSW implementation, each new generic invocation of the Works Manager and File Package must also establish a processing context for the new request. This too adds to the CPU demand associated with linkage, although we have no estimates for its extent in general.

Thus, for all of these factors, about 38% of the processing demand is associated with component linkage. Table 1 shows that the Works Manager CPU demand represents around 32% of the total processing demand to support file name lookup, access control and related functions, and that the File Package CPU demand represents about 29% of the total to support file copying and related functions. Thus there are three relatively large, roughly equivalent CPU demands which comprise the entire operation (99%).

We believe that there are three different factors which at least partially explain these three extensive NSW CPU demands relative to similar functions within single site, conventional, native host operating systems. Linkage CPU demand is most related to structural differences between NSW and the native host system. File Package demand is most related to functional or conceptual system differences. Works Manager demand is most related to internal component organizational differences. We examine each of these individually.

NSW is structurally very different from a native host operating system in which typically each major component is interconnected through some sort of efficient, although sometimes very complex, subroutine linkage. The NSW implementation has each major component as a separate timesharing job. The system design attempts to handle the most complex situation where a user connected to one machine, might be running a tool on another machine, which references a file on yet another machine. A single implementation strategy was pursued which only handled the most general case. This is not unlike a conventional operating system, except that linkage between the logical entities is typically via unprotected subroutine call at machine instruction speeds instead of by messages at communication speeds.

When the tool and file are indeed on separate physical hosts, there is no alternative to communication oriented linkage, which is the major cost for the added functionality of non-local

file referencing. However, for local data references, when the native operating system become an overhead benchmark for functionally equivalent services, an optimized strategy which avoids or reduces the interprocess communication linkage cost seems to be required.

If all of the linkage costs were eliminated, NSW file transaction processing would still not compare favorably with the local host equivalent function. Part of the reason is a functional difference between the NSW and a local host versions of the similar operation. In the native host environment, access to a file opens a more or less direct path to the stored file data. NSW does not currently support such direct access to its files. Instead, it supports the notion of a workspace copy of the data in the original file. This means that, in general, whenever an NSW file is referenced, a copy of it must first be made before completing the tool's file access request. Further, since the file may originate on another type of host, the most general case includes network transmission and translation to generate the workspace copy. The copy model for the NSW file system was motivated by the notion that copy of access is appropriate to many aspects of software production and by an attempt to achieve uniform NSW file system semantics over a distributed file system, while continuing to use local file access methods. The burden of the file copy related overhead falls to the File Package, and accounts for at least some of its extensive CPU demand.

The Works Manager performans the catalog lookup and access control check for NSW objects (files, tools). Native host file access implementations provide similar functions without the large CPU demand exhibited by the Works Manager. As noted previously, the main difference we see between the Works Manager implementation of these functions and the local host implementation is in the organization of the data which represents the catalog, and the procedures used to access it. The Works Manager uses a general purpose, associative information retrieval data base system as the basis for the resource catalog implementation. Although this approach provides great flexibility and supports a powerful user interface to the cataloged objects, it is not as efficiently encoded or as inexpensively accessed as native host conventional filing systems.

## 10.4 Performance Enhancements

The NSW instrumentation and performance evaluation effort provided a number of insights into various optimization strategies. Below we list some of the steps that were taken over an extended period which were in some way related to improving performance. In some cases performance enhancement was a by-product of an effort which was originally targeted for other purposes.

- Reduced Component CPU and Memory Demand

Using the results of the instrumentation mentioned earlier, each component underwent an internal optimization, reducing CPU demand dramatically, but with somewhat less successful results in reducing paging demand. This may reflect the inadequacy of automated tools for dealing with dynamic memory allocation issues.

- Host Hardware and Operating System Upgrade

  The major support host for NSW was upgraded from a DEC KA-10 running TENEX to KL-20 running TOPS-20. This upgrade improved the CPU performance approximately by a factor of two, and served as the basis for larger memory systems which can more effectively support the large virtual memory demands of NSW.

- BCPL Compiler Improvements

  Some of the major NSW components (Works Manager, TOPS-20 File Package) are coded in BCPL. Recent improvements to the BCPL compiler include an optimized code generator, as well as support for in-line assembly code. NSW software is now being adapted to use the enhanced compiler. Benchmark tests indicate a code size reduction of about 15% using the new compiler.

- NSW Protocol/Scenario Enhancements

  A number of NSW Scenarios were modified in an attempt to provide the same functionality more efficiently. Most notable here is the optimized placement of system functions in an effort to reduce communication overhead and intercomponent transactions. The expense of moving these functions has predominantly been in re-implementation costs. This is an on-going activity.

- System Architecture Enhancement

  An implementation of the UNIX Front End has recently been introduced into the NSW configuration, providing extensive on-site computing capabilities, and reducing communication cost and overhead.

- System Design Enhancements

  The design of certain parts of the system functionality were modified/enhanced in part to support optimized performance. Notable improvements here include the extended lifetime of workspaces to support multiple, consecutive tool invocations without deallocation and subsequent re-allocation, and modified object lookup rules which can be supported by an optimized catalog implementation.

NSW version 6.0, which incorporates a number of these performance enhancements, is now under development.

## 11. OPERATIONAL ISSUES

One of the phases of the NSW project was a period in which a number of tools and procedures were developed to improve the operability of the system. As with most such experimental projects, initial procedures for operating and configuring the system were ad hoc. The multiple heterogeneous host environment also contributed to the level of difficulty in operating, testing and debugging the system. This section discusses a few of the tools and mechanisms which were developed in attempting to transform the NSW system into a product in preparation for the AFLC technology demonstration.

### 11.1 Global Configuration File

Many of the NSW components were developed by different organizations and by different programmers within the organizations. Especially across the heterogeneous components (i.e. TBH software for the various ARPANET hosts) the style of controlling operational parameters varied greatly. These parameters would typically include site dependent data such as directory names used in the implementation, switches to control debugging aids and logging, and parameters for such variables as timeout intervals and well-known socket addresses. In some, cases component parameters such as these were built into the programs requiring recompiliation to change them. In other components they were initialized using an interactive dialog the

first time the component was run. Still others used private initialization files with private formats for encoding their choice of relevant parameters.

To simplify the operation of the system when it began to be operated by a group independent of the system developers, a global configuration file was defined. The global configuration file was intended to define parameters which would be globally interpreted by the various components (e.g. logging control switch), as well as to serve as a repository for all components and site specific parameters in addition to the globally defined parameters. The global configuration file is a specially formatted text file containing all of the parameters relevant to an NSW configuration. The file is designed to be maintained at a central site and broadcast whenever it is changed to each of the hosts of the configuration. At system startup, each component searches the local copy of the configuration file to find both the global and private parameters which have been set for this incarnation of NSW service at the host. In this manner, an NSW operator can easily change the operational characteristics of the NSW system using available text editing and file transport services.

## 11.2 Fault Logger

In a situation similar to that described above, each of the NSW components and hosts had private mechanisms for recording

indications of faults and errors detected during their operation. Since many of the components were run as unattended tasks which were not in contact with a user, the predominant method for recording relevant parameters of detected failures was to write the failure report into a locally maintained fault log file. Depending on the resources available locally, this file was sometimes overwritten by failures in subsequent NSW incarnations. This distributed approach to failure recording made it quite difficult for NSW operators to quickly detect and repair or obtain help in repairing system problems.

To remedy this situation, an additional NSW system component, known as the Fault Logger was defined and implemented. The Fault Logger is the NSW component which collects fault messages from other NSW components, and records and displays them for the system operator. Fault messages are structured records of errors detected by the system components while the system is running. They indicate the nature of the fault detected, and include other parameters and debugging aids that can be used to help isolate the problem.

When a fault message is received by the Fault Logger, it is recorded in a master fault repository. At the same time a copy may be printed on a hard copy log, while additional copies may be deposited in other special repositories or given to special processing programs. All actions other than depositing the message in the central repository are conditional, and are under

the control of system operators. The conditional processing is controlled by filters which are used to scan the fault report messages for content criteria matching the filter. For example, a filter can be used to cause only fault report messages above an operator specified priority level to be displayed on the operator's terminal.

To complement the fault message repositories, there is an "off-line" retrieval system for searching, sorting and collating individual fault reports or collections of faults based on their contents. In addition, there is an on-line operator interface to the Fault Logger, enabling the operator to create, modify and install appropriate filters used to control optional processing of arriving fault messages.

### 11.3 NSW Bug Report Tracking Tool

The NSW project involves many people working for different organizations dispersed geographically across the United States. This project structure limits highly interactive communication between project personnel which is crucial to the timely completion of many project activities. This has been reflected in the amount of time that was needed to achieve design and protocol consensus and in difficulties with component integration. Additionally, the identification, reporting, tracking and correction of bugs became a suprisingly difficult task. This was due in part to the complexity of the system and

in part to the limited communication paths between project members. This problem was addressed by the development of a tool to record and track bug reports. The name of the tool is MONSTR, which stands for Monitoring Software Trouble Reports (STR).

Driven by an embedded protocol, MONSTR coordinates and supports the activities of people in various organizations as they report, acknowledge, classify, analyze, and repair bugs. Thus, it is a tool for maintaining contact among all personnel concerned with particular software repair efforts.

There are three main groups which MONSTR serves to interconnect. The first group is composed of people report bugs, deficiencies, anomalies which may or may not be bugs, and documentation errors. These people do not generally get involved in fixing bugs. NSW users are included in this group.

The second group is involved in fixing bugs as well as reporting them. This group includes the NSW operators and developers.

The third group includes NSW managers, such as the project sponsors and the head of the Architecture Control Contractor (ACC) organization. These people generally do not report or fix bugs. Instead, they monitor the work-flow of those who do.

MONSTR has facilities for generating and transmitting STRs, splitting and merging STRs, cancelling and modifying STRs, and classifying and retrieving STRs. It includes well-defined

internal protocols for STR transmission from organization to organization to initiate the identification, fixing, and installation of the software repairing the error. MONSTR also maintains the current status and a history for each STR.

MONSTR has been installed as a tool in the NSW system so that users can easily report system problems while on-line, and can subsequently check on the status of the effort to fix the problem.

## 12. CONCLUDING REMARKS

At this time NSW must still be viewed as a prototype system implementation. It is only now undergoing significant testing by actual users. It's major impact to date has been the experience gained both in exploring functionality which is useful in a distributed operating system context and in evaluating specific approaches toward achieving this functionality. When viewed in this light, and in terms of the additional experience gained in uncovering and facing a wide variety of distributed system issues, the project has been extremely successful. We have had the opportunity to build, assess and rebuild certain aspects of the system functionality and structure. The NSW Technology Demonstration should prove useful in assessing some of the premises on which the NSW was built.

Defining, designing, implementing and operating the NSW system has been a difficult undertaking from both a technical and non-technical viewpoint. The NSW project was perhaps the first to focus on the whole set of issues, spanning the spectrum from design to operational considerations, for a system of this type. The problems of dealing with a completely distributed system environment are new, and the problems of dealing with extensive heterogeneity in such an environment are hard.

In part, that heterogeneity and the desire (and necessity) to have experts in the various constituent systems participate in the system design process led to a project organizational

structure which mirrored the underlying computing environment: geographically dispersed project team members using relatively low bandwidth communication paths to coordinate their activities. This type of project organization was not particularly well suited for general purpose system building, for which we believe consistency, coherency and integration of the system parts is the overriding consideration. A geographically distributed project organization makes it more difficult and time consuming to achieve consistency and a clarity of the product. However, a beneficial side effect of the distribution of the project organization is that system concepts have 'a priori' been exposed to and influenced by a number of different system design perspectives and philosophies.

Another non-technical issue that had significant impact on the project was the lack of coordinated administrative control over the project's computer resources. Although NSW provides a logically centralized administrative view of the system, its computer resources are obtained from a number of administratively distributed and autonomously operated machines connected to the ARPANET. Two aspects of this arrangement were especially troublesome. One was the frequency of host system modifications and upgrades. Because NSW is itself an operating system running on top of existing operating systems, parts of it were especially sensitive to relatively small changes in the underlying systems. In addition, many of the software tools supported by the underlying hosts are an important part of the overall NSW system.

The new releases, bug fixes, etc. for a given system or tool were seldom coordinated with NSW project plans or with each other. As a result, there are many more disruptions of this type for the system maintainers and users than had the same resources been under a single administrative control and hence subject to more global coordination.

The disruptions due to new releases of software tools were minimized by designating particular versions of tools as NSW versions supported by private copies coordinated where necessary among collections of participating hosts. This had the effect of providing global update control within the set of NSW hosts, but at the expense of timely access to the new improved host supported maintenance releases.

The other negative aspect of multiple administrative control of hosts which had a significant effect was that multiple versions of the same operating system (e.g., TOPS-20 Version 4 and TOPS-20 Version 1) needed to be supported simultaneously by NSW software. Worse still was the prevalence of local independent customization of the operating system at each site which necessitated a variety of special case situations to be individually uncovered, handled, tested and supported. These situations were a constant and heavy drain on project manpower.

In conclusion we briefly note a few key technical issues which emerged as a result of our experience with building the NSW system, some of which have been partially addressed in newer versions of the system.

1. Visibility of Distribution

This issue relates to the degree to which the distributed nature of the underlying system is visible to and controllable by the individual user. There is a continuum of designs possible. At one extreme is complete invisibility (transparency), where a user need not be aware of the network operations or network host systems used on his behalf, and in fact cannot directly exert any control over the manner in which they are used if he is aware. At the other extreme is complete visibility where the user must directly exert control over the selection of resources to service his requests. The NSW project started out decidedly on the transparent side of the spectrum. Over time it has slowly but surely moved toward the less transparent side in order to provide more user control of resource management decisions and to accommodate resources which could not have been adequately handled under a policy of maximum network and host transparency. Some reasons for this shift in approach are the following:

- Due to the differences in performance and reliability of local and remote operations (at least in an ARPANET-like environment), users had to develop working models of how the system was put together anyway, in order to effectively use the system when conditions were non-optimal (e.g. in failure and high load situations).

- An understanding of various system components and hence the underlying structure of the system is sometimes required for a user to take full advantage of various optimizations added to improve the performance of the system.

- The high cost of introducing new hosts and new resources into the system limited the scope of the available resources. In an attempt to achieve a more "complete" system by providing access to more resources without having to rebuild or completely integrate the desirable resources, lesser degrees of integration were defined. This obviously led to greater visibility of the underlying milieu.

- Related to the above point, different levels of effort were applied to the integration of "fully" participating hosts. In addition, the integration of these hosts and their services by necessity took on a distinctively local flavor (i.e. strongly related to the environment in which they

were originally developed). This also led to a need for increased awareness of the individual characteristics of the various resources.

2. Granularity of Objects

One of the key issues in the design of any distributed operating system is the granularity of the objects provided by the system. In a system like NSW, one could choose abstractions ranging in size from an entire host, to a file, down through a page of data, and even smaller as a basis for achieving the required logical integrity of the system. For NSW, the chosen grain of an entire file and an entire application program (service) seems to be appropriate, given the dominant role of existing large main frame computers. However, other environments which may include less complex host operating systems, faster communication media and machines, or applications built specifically for the distributed system environment may profit from a different unit of granularity in achieving logical uniformity. In fact, later versions of NSW have been enhanced to include larger grain abstractions, such as an entire host operating system, partly to reduce the effort needed to integrate resources into NSW in an acceptable manner, and partly as a means of providing the raw performance of the stand-alone system for those users to whom this was appropriate.

3. Duration of Context

A related issue concerns the longevity of the bindings which are established during the course of using the system. The number and frequency of the dynamic bindings needed to support user activities has a profound influence on the performance of the system. There are, however, tradeoffs between flexibility, performance and complexity which need to be tailored to the specific environment. Early versions of NSW established new file-to-host bindings and user-to-service host bindings with each NSW tool invocation. This was the most universal and flexible binding model. However, when successive tool sessions run on the same host and access the same files, re-establishing the same bindings is unnecessarily expensive. Later versions of the system achieved performance improvements in these situations by such techniques as extending the lifetime of some of the critical bindings through the refinement of the workspace concept and caching certain objects. Because the overhead associated with establishing bindings in a distributed

system is often more complex and therefore more expensive than in a conventional system, and because there are typically many more binding decisions in a distributed system, this is a very critical distributed system design area.

4. Multiple Copy Objects

The potential for various types of replication and redundancy in a distributed system architecture introduces a number of issues relating to the semantics of system objects. For example, what level of "sameness" is required amonvg similar objects for them to be considered instances of the "same" object, as opposed to different objects with some special relationship? NSW started out by considering relatively loosely coupled instances of similar objects were to be the same object (e.g., the same tool on different hosts, or workspace file copies and the original). Later versions of the system tended to decouple the association at lower levels while providing means to treat a set of similar objects as a single object for some classes of operations at a higher level. The issues here can be quite complex in a distributed, heterogeneous environment where subtle and not so subtle differences in environment may introduce differences in the objects at some level of abstraction, and result in automatic conversion to overcome many (but not usually all) of these differences. Determining what role the system should play in supporting these special relationships is an important design issue.

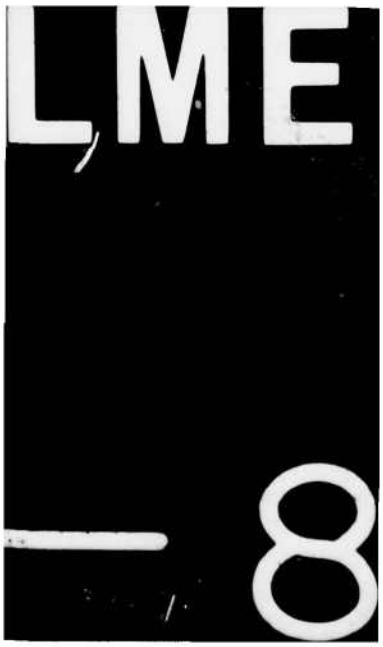
5. Migrating System Functions

The experience with the development and evolution of the NSW system has led us to two important observations regarding the construction of such distributed systems. One is that it is extremely hard to test, debug, and to isolate problems in such systems. The development of testing and debugging methodologies and tools seems to be an important area for research. Second, it seems inevitable that, over time, functions which are implemented within one component of the system will (statically) migrate to other components. Such migration may be motivated by performance enhancement based on locality, changes in technology, an expanding role for a component, or any of a number of other reasons. All too often, this requires total re-implemention of the function in the new context. The process abstraction in NSW that is supported by MSG was extremely effective in allowing complete

reconfigurability at the process level. However, because processes and interprocess communication were expensive this flexibility extended only to decomposition of the software architecture into major system components. Moving smaller units of functionality from one host to another or from one process to another meant sometimes excising a function from one place and always reimplementing it elsewhere. The NSW project repeatedly faced this situation. A standard system wide implementation language (not a possibility when the NSW project started) might be a start toward supporting this type of functional migration. However, much more is needed to make this migration even semi-automatic such as re-establishing all of the distributed bindings to the migrated function. This area also seems to be one in which more research and development effort could profitably be applied.

# *MISSION*
# *of*
# *Rome Air Development Center*

*RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence ($C^3I$) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.*


LME
8