

AD-A185 967

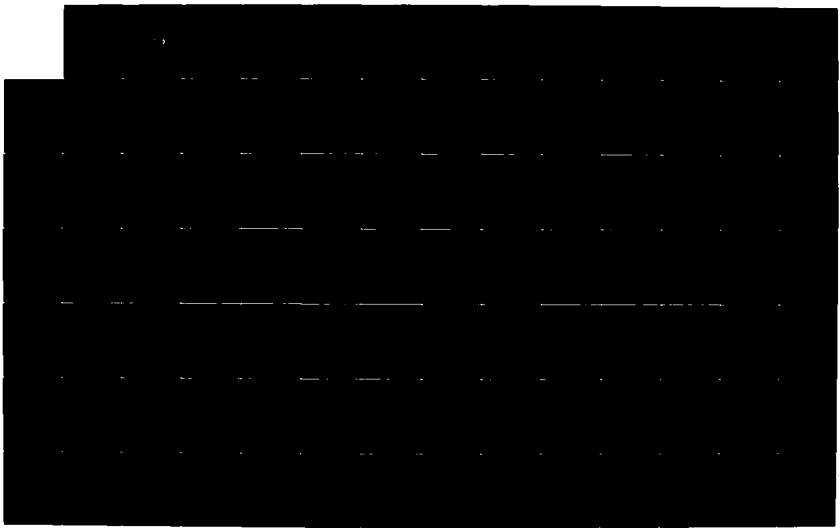
NSW (NATIONAL SOFTWARE WORKS) PERFORMANCE ENHANCEMENTS
(U) BOLT, BERANEK AND NEWMAN INC CAMBRIDGE MA
R E SCHANTZ ET AL. JAN 81 BBN-4600

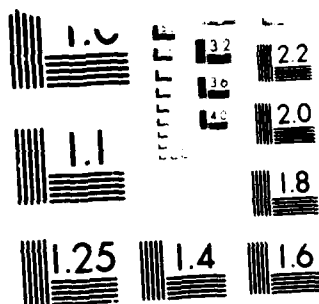
1/4

UNCLASSIFIED

P G 12/5

NL





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

FILE COPY

Bolt Beranek and Newman Inc.



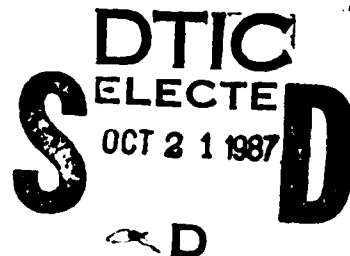
AD-A185 967

Report No. 4600

NSW Performance Enhancements

Final Report

Richard E. Schantz and Robert H. Thomas



January 1981

Prepared for:
Rome Air Development Center and
Advanced Research Projects Agency

Approved for public release; distribution unlimited

87 10 8 024

Report No. 4600

NSW PERFORMANCE ENHANCEMENTS
FINAL REPORT

Richard E. Schantz
Robert H. Thomas

January 1981

Prepared by:

Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, Massachusetts 02138

Prepared for:

Rome Air Development Center
Advanced Research Projects Agency



Accession For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Spec
A-1	

TABLE OF CONTENTS

	<u>Page</u>
1. Project Summary	1
2. Task Activities	3
2.1 Implementation and Maintenance Support	3
2.2 Modifications to TOPS-20 and TENEX Operating Systems	4
2.3 Installation of Selected TOPS-20 Programs	4
2.4 Technical Support for NSW Contractors	5
2.5 NSW Performance Evaluation	5
2.6 Extension to Capabilities of TOPS-20 NSW Components	6
2.7 Provide NSW Mail Service	9
2.8 Provide a System-Wide NSW Fault Logger	10
2.9 Analyze NSW System Requirements	10
2.10 Develop a Functional Specification	10
2.11 Detailed Design for Highest Priority Enhancements	11
2.12 Participation in the Nsw Technology Demonstration	11
3. Future Developments	11
4. Software Developed Under this Contract	12
5. Documents Prepared Under this Contract	13
APPENDIX A. NSW Concept of Operation	A
APPENDIX B. NSW Functional Specification	B
APPENDIX C. NSW Enhancements Detailed Specification	C
APPENDIX D. NSW System Performance Evaluation	D

1. Project Summary

This report is the final report for RADC/ARPA Contract No. F30602-79-C-0110, ARPA order No. 3686, entitled "NSW Performance Enhancements." The NSW Performance Enhancements project was part of the National Software Works (NSW) program sponsored jointly by ARPA and the Air Force. The objective of the NSW program is to develop a network operating system which provides an effective environment for DoD software production. The general objectives of this project were to develop and maintain the software necessary to integrate the TENEX and TOPS-20 systems into NSW as tool bearing hosts (TBHs) and to evaluate and improve overall NSW performance. The specific tasks for this effort as enumerated in the statement of work were:

- Implementation and Maintenance support for TOPS-20 NSW Software,
- Implementation and maintenance support for TENEX NSW software,
- Modifications to TOPS-20 and TENEX Operating systems to support NSW system software,
- Installation of selected TOPS-20 programs as NSW tools
- Technical support for NSW contractors,
- NSW performance measurement and evaluation, and the design of performance improvements,
- Extensions to capabilities of TOPS-20 NSW components
- Provide NSW Mail Service,
- Provide a system wide NSW Fault Logger.

During the course of the two year effort, the overall direction of the NSW project was altered somewhat to reflect the identification and selection of the Air Force Logistics Command (AFLC) as the target group for an NSW technology demonstration. As a result, four additional tasks were added:

- o Develop a set of system requirements which an NSW-like system must service to adequately support various classes of user communities.
- o Develop a functional specification of an enhanced system design which could meet those requirements through evolution of the current NSW system.
- o Provide a detailed design for a selected set of those system enhancements directly oriented toward supporting AFLC use of NSW during the technology demonstration.
- o Participate in the NSW Technology Demonstration Working Group to help select meaningful experiments in utilizing NSW and to help ensure that NSW evolves in ways which can best support those experiments.

There are two significant aspects to the NSW effort expended under this contract. One aspect relates to the operational support of the current NSW system. As a result of two major NSW system releases during the course of the effort, system reliability has been enhanced, and component release, document update, bug reporting and program maintenance procedures are now well established in anticipation of an active user community during the technology demonstration phase of the project.

The other more technically oriented significance of the current effort relates to the evaluation of NSW technology and developing a larger framework (i.e. the functional specification) for its enhanced application. The effort also marks the beginning of the evolutionary transition toward a system which we anticipate will be better able to support the goals of the AFLC user.

In the following section we discuss some of the activities which were undertaken during the course of the contract for each task, and where pertinent indicate the status of the task. Appendices A, B, and C reproduce documents covering the NSW Concept of Operation, the NSW Functional Specification and NSW Enhancements Detailed Design, which were prepared under this effort. Appendix D reproduces a report evaluating current NSW system performance, which was also prepared under this effort.

2. Task Activities

2.1 Implementation and Maintenance Support for TOPS-20 and TENEX NSW Software

During the course of the two year effort, there were two major releases of the NSW system (version 4.0 and 5.0). A later section will briefly mention a few of the component enhancements which were included in these releases. For each of the modified components there was a high level of maintenance activity associated with releasing the system. This can be attributed partly to the complexity of the NSW project organization and

the system itself, partly to the fact that we were concurrently supporting three different versions of the operating system environment (TOPS-20 release 101 B, TOPS-20 release 3, and TENEX version 1.35) on three differently configured hosts, partly because there are significant differences in the various NSW system configurations (e.g. Debug System, Development System, etc.) and partly because of the limitations of unit testing new component releases. During these releases, procedures for automated bug reporting and tracking were introduced and subsequently modified to suit the needs of the current NSW project. Currently, most of the software problems reported to us have been corrected and updated software has been released to the operational system. There are a few problems, mostly in the area of encapsulated tool interfaces, which will be corrected as part of a projected NSW release 6.0.

2.2 Modifications to TOPS-20 and TENEX Operating Systems to Support NSW Software

No additions to the TOPS-20 or TENEX operating systems were required for supporting the NSW software on those systems.

2.3 Installation of Selected TOPS-20 Programs as NSW Tools

Due to the changeover to TOPS-20, and the significant program interface modifications associated with the changeover to TOPS-20 release 3 or 4, all tools previously installed in the TENEX based NSW system had to be reinstalled and retested for the TOPS-20 based system. In addition, a number of new

TOPS-20 tools were added: ECL (Extensible Computer Language, an innovative programming environment developed at Harvard), ADA-PARSE (a utility program to check the syntax of ADA programs), and SRCCOM (a document and program maintenance tool which identifies the differences between two versions of a file). All of these tools have been installed via encapsulation mechanism which means that their operation in the NSW is practically equivalent to their operation directly through TOPS-20, without modification to the tool.

2.4 Technical Support for NSW Contractors

There were two major areas in which we provided continued support of the design and development activities of other NSW contractors. The first area was in alerting other organizations to the potential impact on NSW software of new TOPS-20 releases. The second area was in working with UNIVAC personnel in developing an approach to connecting a UNIVAC 1108 to the ARPA network and in designing host software to make a UNIVAC system available as an NSW tool bearing host.

2.5 NSW Performance Evaluation

A number of experiments to measure various performance characteristics of the NSW system were designed and carried out during the course of the contract. These experiments helped to provide a more in-depth understanding of the performance of the individual NSW components, and the behavior of the NSW system as viewed from the users perspective. The observed system performance is a complex relationship between individual

component performance, inter-component relationships and relationships between the components and the host operating system. The experiments and their results were reported in two documents issued during the course of the contract: "A Performance Analysis of the NSW System", BBN Report Number 3847, March 1979, and "NSW System Performance Evaluation", BBN NSW Note No. 29, February 1980 (reproduced as Appendix D of this final report). Partly as a result of these evaluations, we were better able to formulate a number of system performance enhancements for inclusion as part of the functional and detailed design specifications which were developed.

2.6 Extension to Capabilities of TOPS-20 NSW Components

The TOPS-20 MSG component was enhanced in a number of areas during the course of this contract. These included modified MSG host addressing to support ARPANET extended leaders, better management of the use of timeouts, and more robust handling of ARPANET direct connections. The major implication of these latter two efforts is the vastly improved reliability and fault recovery characteristics of TOPS-20 MSG. In addition, a new MSG maintenance and diagnostic tool was developed to automate parts of the checkout procedure for new MSG releases.

The MSG enhancement to support ARPANET extended leader addressing was the most pervasive of the improvements, since it required an update to the specification and system-wide changes.

Extended leader addressing extended ARPANET host addresses from 8 bits (addressing a maximum of 63 IMPS) to 24 bits. In addition, internetwork addresses have been defined to include an additional 8 bits identifying a particular network (i.e. internetwork addresses are 32 bits wide). The MSG specification provides only 16 bits for host addressing, which had been used to hold the old-style 8-bit ARPANET host address. One constraint we adopted in supporting the addressing enhancement was to minimize the extent of the modifications on the NSW components as a whole. Because of this we chose to handle the extended addressing through a mapping between the most pertinent parts of the extended address and the current 16 bit MSG address field. This change has been implemented for TOPS-20, and the current NSW system supports a host which requires extended addressing in order to be accessible. Further details on the nature of the problems encountered and the complete specification for their solution can be found in BBN NSW Note No. 31, August 1980.

The major extensions to the TOPS-20 Foreman component were in the areas of automated workspace management, extended reliability functionality including allowing the automated continuation of interrupted tool sessions, and tool encapsulation interface extensions to allow users more control over workspace file selection and processing. The workspace management extensions are significant improvements in the operational support for NSW on TOPS-20 and include the dynamic creation of tool workspaces,

the "archiving" of old, saved sessions, and enhanced operator control of workspace handling.

Workspace management is primarily concerned with assigning a private operating context (a workspace) to a tool invocation request, and then managing that resource on behalf of the tool/user until the tool session is successfully completed. Prior to our workspace management enhancements, if a TBH would crash, workspace files for users active at the time would be permanently preserved in the workspace to allow later recovery of the tool sessions' interim results. This made the workspace resource unavailable for reuse until either the saved session was successfully rerun, or the operator interceded to discard the session. To alleviate this problem, we have implemented software to automatically move saved tool sessions from workspaces to other areas of secondary storage, whenever workspaces become unavailable. This workspace "archiving" facility also includes automatic retrieval of archived sessions when a re-run request which references the saved session is received.

To further augment workspace management capabilities, we have incorporated a new TOPS-20 feature, sub-directories, into the TOPS-20 Foreman. Using this feature, the Foreman can dynamically create and deallocate new workspace directories as needed to handle the current NSW tool load. This aspect of workspace management is automatically handled by TBH software based on guideline parameters provided by the system operator as part of the system configuration data.

The inclusion of these workspace management extensions and operator tools for controlling them, fully automates the procedures for workspace management, lessening the need for operator intervention, reducing tool initiation failures due to unavailable TBH resources, and permitting the long-term storage of saved tool sessions. All of these enhancements are provided as local optimizations of the current system specification and remain transparent to the NSW user and all NSW protocols. Complete details on this topic can be found in "Workspace Management in the TOPS-20 Foreman," BBN NSW Note No. 30, June 1980 (also available as Appendix A of the "The Foreman" Providing the Program Execution Environment for the NSW", BBN Report No. 4587).

In support of these system enhancements, we have revised and released new versions of the appropriate System/Subsystem Specification, User, and Program Maintenance Manuals for the effected NSW Components.

2.7 Provide NSW Mail Service

The effort to support network mail services within NSW was subsumed under the system redesign effort. The NSW Functional Specification (Appendix B) provides a preliminary design for NSW mail services. The selection of the UNIX Front End as the NSW interface host for the AFLC technology demonstration will significantly impact the implementation approach when the mail support task is continued in subsequent NSW development activity.

2.8 Provide a System-Wide NSW Fault Logger

During this contract, we completed the design, implementation, test and operational installation of an NSW Fault Logger system including an operator utility for controlling its use. The Fault Logger is the NSW component which is the designated recipient of fault reports from all other NSW components. It provides sophisticated support for logging, displaying, sorting and forwarding fault messages under the control of the NSW system operator. We have prepared and released a System/Subsystem Specification document, as well as a User and Program Maintenance Manual for the Fault Logging subsystem. Refer to that documentation for a complete description of the Fault Logging system.

2.9 Analyze NSW System Requirements

Under this task, we analyzed various sets of requirements which an NSW-like system should meet for different user communities. The results of this task are reported in Appendix A.

2.10 Develop a Functional Specification for an Enhanced NSW System

This task involved a complete functional specification for an NSW-like network operating system which could meet the requirements set forth in the preceeding task. This task was intentionally linked to the existing NSW architecture as a starting point for specifying functional enhancements. The complete functional specification is provided in Appendix B.

2.11 Detailed Design for Highest Priority System Enhancements

From the collection of functional enhancements specified as part of the previous task, a subset of high priority enhancements for supporting the AFLC technology demonstration was selected. Under this task, we have developed detailed design specifications for a complete set of system enhancements which are to be incorporated into an NSW release 6.0. The most current draft of this design specification is included as Appendix C.

2.12 Participation in the NSW Technology Demonstration Working Group

To carry out the NSW technology demonstration, a working group consisting of Air Force Systems Command and Air Force Logistics Command Personnel was formed. Since its inception, we have been a principal consultant to that group in a variety of ways. We have attended working group meetings in an attempt to learn some of the AFLC problems which may be addressed by networking and NSW technology. We have participated in the development of and helped to specify the scenarios of use for the NSW system during the technology demonstration. Finally, we have analyzed the objectives of the technology demonstration and organized a list of system enhancements tailored to the intended patterns of AFLC system utilization.

3. Future Developments

At this time, an NSW follow-on effort is being contemplated by the system sponsors. The intent of the effort would be to

provide the NSW system enhancements designated as high priority for AFLC use of the system, and to continue to support the operational NSW system during the technology demonstration. This effort involves implementation of the enhancements specified in Appendix C, as well as the design and implementation of additional high priority enhancements. It also includes the installation and test of specific software tools applicable to the AFLC mission objectives, and continued participation in the NSW Technology Demonstration Working Group activities to help ensure a successful experiment.

4. Software Developed Under this Contract

- o New versions of TOPS-20 MSG up to and including the NSW 5.0 release.
- o Initial version of MSGTEST, a diagnostic tool for testing TOPS-20 MSGs.
- o New versions of the TOPS-20 Foreman up to and including the NSW 5.0 release.
- o New versions of MKCOM, the operator utility for managing and monitoring TOPS-20 tool bearing host workspaces.
- o New versions of the TOPS-20 NSW Front End Dispatcher and NSW Root programs used to remotely initiate and monitor TOPS-20 NSW Front End jobs.
- o New version of the NSW Fault Logger, for receiving and logging NSW fault reports.
- o Initial version of FLOPER, the Fault Logger operator utility for monitoring and controlling the operation of the FL.

- o Initial version of FLTEST, a tool for simulating fault traffic for testing new versions of the Fault Logger.
- o Software to record and process the virtual memory page access referencing behavior of a TOPS-20 program (to try to improve it locality of references).
- o Software to automate NSW user session oriented performance testing using typescripts.
- o Various TOPS-20 based performance test and measurement programs.
- o Software to implement protocol for controlling shared access to test NSW configurations.

5. Documents Prepared Under this Contract

- o "A Performance Analysis of the National Software Works", BBN Report No. 3847, March 1979.
- o "MSG TOPS-20 User Manual", BBN Report No. 4701, May 1980.
- o "The Foreman: Providing the Program Execution Environment for the NSW, System/Subsystem Specification", BBN Report No. 4587, January 1981.
- o "TOPS-20 NSW Foreman Program Maintenance Manual", BBN Report No. 4093, May 1980.
- o "NSW Fault Logger System Specification", June 1980.
- o "NSW Fault Logger User's Manual", June 1980.
- o "NSW Fault Logger Maintenance Manual", January 1981.
- o "Protocol Modifications to Support Improved Performance for Tool Use of the NSW File System", BBN NSW Note No. 27, February 1979.
- o "A Note on the NSW Global Configuration File", BBN NSW Note No. 28, February 1980.
- o "NSW System Performance Evaluation", BBN NSW Note No. 29, February 1980 (included as Appendix D of this report).

- o "Workspace Management in the TOPS-20 NSW Foreman", BBN NSW Note No. 30, June 1980.
- o "MSG Host Addressing", BBN NSW Note No. 31, August 1980.
- o "NSW System Concept", included as Appendix A of this report.
- o "NSW Functional Specification", included as Appendix B of this report.
- o "NSW Enhancements (version 6.0) System/Subsystem Specification", included as Appendix C of this report.

NSW CONCEPT OF OPERATION

Prepared by
Bolt Beranek and Newman Inc.

Originally Issued November 1979

Appendix A

INTRODUCTION

NSW is an operating system designed to support users in a computer network environment.

In some respects NSW will appear 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 adequately 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, memos) 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.

CANDIDATE SYSTEM MODELS

Given the nature of the network, the host environment and the users, there are at least three somewhat different perspectives on what an NSW system might be.

1. NSW as a monitor.

As a monitor NSW would be a completely self contained network operating system. After logging into the NSW monitor a user would work entirely within the NSW environment and under NSW control. All accessible host resources would be uniformly integrated into NSW and uniformly accessible from it. NSW would implement a distributed network filing system and an environment for program execution. Programs would be uniformly accessible regardless of the hosts that housed them, they would execute against the NSW file system, and data files referenced by programs would be uniformly accessible regardless of their location relative to the programs. The sharing function would be accomplished through the file system, and the ability to access remote services would be provided through the program execution environment.

The other two views of NSW represent somewhat more limited and less sophisticated system possibilities. Although they are less ambitious, each provides useful capabilities.

2. NSW as a sharing and communication medium.

Here the principal function of NSW would be to maintain a repository of files for sharing among users. In addition to providing storage for files in the repository, NSW would provide a variety of bookkeeping services. These services would include file cataloguing functions that implement a host independent global file name space for files in the repository, and access control functions to permit flexible controlled sharing of files in the repository. Users might prepare files outside of NSW and deliver them to NSW when ready to be shared. NSW would provide a convenient way to move sharable files into the repository and to move copies of them out of it.

3. NSW as a service access mechanism.

Here the principal function of NSW would be to provide convenient access to services distributed among network hosts. The user would enter NSW and invoke a desired service by name. After performing the access operations necessary to place the user in direct contact with the service NSW would become transparent, enabling the user to interact directly with the service. Services would operate in their native host environments. Accessible services would be registered with the NSW, enabling information necessary to access the service to be catalogued.

SYSTEM ENVIRONMENT

The network is geographically distributed. Communication between the host computers is generally limited to tens of kilobits/second (or less). The host computers connected to the network are of different manufacture, run different operating systems, and have widely varying computational power and storage capacity. Some hosts provide interactive timesharing service, some provide batch processing capability, some serve to provide users terminal access to the network, and still others operate single user systems. Some hosts serve to provide a variety of these functions.

Although the initial implementation of NSW will be supported by the ARPA network, the system concept is valid for other network environments, including internet environments. An important system design goal is for NSW to be readily transportable from the ARPA network to other comparable networks, and to be easily extendible from the ARPA network into the internet environment.

USERS

Just as there are great differences among the hosts, there are great differences among the users of the network in terms of their requirements, preferences (for computer system features) and levels of sophistication (in using and interacting with computer systems). While it is difficult to characterize the users in detail, it is possible to identify two general areas where an NSW operating system would facilitate use of network resources.

One area is support for collaborative efforts among users who may be distributed as the hosts are, or among large numbers of users. The collaborative efforts of interest range in scope from the preparation and distribution of memos and papers, involving a handful of authors and perhaps many readers, to the production, maintenance and configuration control of sophisticated software systems, involving many programmers, documenters and managers. While the particular functions required to support the various collaborative efforts vary greatly, the fundamental system function required is a convenient means for sharing (files, data, source programs) in a network environment.

The second area is that of providing access to services that are distributed among the hosts of the network. By service we include a wide range of possibilities. Services of interest range from a single program or tool (such as a text formatter or compiler), to a collection of related tools (such as a text editor, compiler, debugger), to an entire operating system (such as Multics or TOPS-20). Again, the particular functions required to provide access to such a spectrum of services varies. However, the fundamental system function required is a convenient means to access remote services in a network environment.

These three system perspectives are, of course, related. For example, a user might prepare an item to be shared through the NSW file repository by using a service accessed through, but not fully integrated into, NSW. To users of services more sophisticated in their ability to manipulate files in the NSW repository, the more closely NSW resembles an integrated monitor. These different system perspectives result from the recognition that it will be impractical to integrate some very useful services fully into NSW without completely reprogramming them. Further, users adequately supported with current tool kits should not be forced into a different monitor environment simply to take advantage of NSW as a medium for sharing in a network environment.

Our task is to determine the extent to which these three perspectives (monitor, sharing medium, access mechanism) can be accommodated within a common NSW framework. To do this, we next consider what users of the system might want or need, and then what we believe would be feasible to implement.

USER REQUIREMENTS

To explore user requirements we identify the needs of a number of different user classes. Some of the classes are characterized by what the users do, whereas others are characterized by how they do it. Thus, the user classes do not represent a partition of the user community. People can be expected to be in more than one user class.

There is a large class of users whose working patterns can be characterized as consisting of periods of independent activity in a private work area separated by periods of interaction, during which the results of the independent activity are delivered for use by others or the results of others are obtained for further independent activity. The periods of interaction occur because the user is part of a community with common interests (e.g., the ARPA research community) or is a member of a project team (e.g., the NSW project). Examples of users in this class include: researchers, whose sharable results are typically reports and memos; programmers, where the sharable results are typically complete programs or modules which are part of large software systems; technical writers, where the sharable results are documents. When necessary to refer to this user class by name, we shall call it user class U1.

The requirements for class U1 users are the following:

- It should be easy to carry out independent activity in a private area. By "easy", here we mean the necessary tools should be available (i.e., part of the "system" they use), readily accessible, and responsive.

- It should be easy to submit or deliver results to the sharing repository, and to withdraw results submitted by others from the repository. Users needing to share results may not be users of the same network host, or users of the repository host(s). To deliver and withdraw repository files users should not be required to deal directly with the network, to be registered users of the repository host(s), or to deal directly with the operating systems of the repository host(s).

In addition, users may want to exercise control over files they have placed in the repository, which other users may access them, and the manner in which they may be accessed. Thus, there should be some form of access control which determines the operations different users are permitted to perform within the repository.

To class U1 users another somewhat smaller class can be added (class U2). Users in this class work primarily within the context of the repository, generally to manage files in it in various ways. Large communities of users of the first class will typically require users of this second class. Examples of users in this class include: "librarians" for files in the repository; system integration personnel for large software systems; software configuration control personnel for large software products.

Class U2 users have somewhat different requirements than those identified above for class U1. These requirements include:

- Means to organize files into related groups and to manipulate such groups of files.
- Tools which can operate on repository files. That is, programs that can run against the repository filing system.

As with class U1 users, these users require an access control system.

The above discussion has focused on use of NSW as a medium for sharing and collaboration. We've seen that to be effective, the medium must provide some monitor-like functions such as a filing system, access control (which implies user registration, identification and authentication (login)), and support for program execution. While more detail could be provided on use of NSW as a sharing and communication medium, we defer doing so at this point.

We now consider the expected user community from a different point of view. We shall assume that NSW supports a filing system and means to move files between it and users' work areas.

Given that some of the hosts are reasonably complete general purpose systems, the needs of many users can be expected to be principally satisfied by a single host (class U3 users). That is, most of their computing needs can be met by their "principal host", and most of their files can be stored on it. These users may, however, occasionally need to use services housed in remote hosts. As suggested above there is a wide range of interesting services; for example, a service might be an interactive text editor, a data base management system, or an entire operating system. For these users NSW could serve as a convenient access path to their principal host as well as to services not provided by the principal host.

After access to the principal host has been accomplished, and while the user is not making use of NSW file repository functions, use of the principal host should be comparable to its use outside of NSW. That is, the user should not see a significant performance difference when working within the principal host environment.

Among other things, this suggests a weak rather than strong coupling between the NSW filing system and the principal host filing system. A useful model here is for the NSW file system to treat the user's file structure on the principal host as a "mountable file structure". The idea is that the principal host file structure would be catalogued in the NSW file system, but that the primary responsibility for managing the files within the file structure itself would be retained by the principal host. This would permit a user's files on the principal host to be conveniently accessible from NSW. The user would refer to such a file by a two part name. One part would be the NSW name for the file structure, and the other part, the name of the file relative to the structure. NSW would interpret one part of the name to determine the appropriate file structure (and host), pass the remainder of the name to the host managing the file structure, and then cooperate with that host to complete the appropriate file operation. We note that the notion of a user having a principal host is compatible with the usage pattern identified above for class U1 users.

Many remote services will operate against user data. Here NSW can provide convenient means for ensuring that the data is accessible to the services.

The term "convenient" has been used to describe both the service access and data access functions NSW can provide. By "convenient" we mean:

- Users need not deal directly with the network or low level network services (such as TELNET or FTP programs). NSW as an access mechanism should incorporate these functions.

- Users need not be subjected to multiple access control checks. NSW knows the identity of all users and should be able to engage in the necessary access control protocols on their behalf.
- Minimal interaction with the host operating system of a service should be required to activate or use the service. (Unless, of course, the service is the operating system itself.) Again, NSW as an access mechanism should activate the service prior to granting the user access to it.

Next, we turn to the nature of the services users might access, and the relation of these services to the NSW system. There is a spectrum of services. To characterize it we first identify the following as a few points on the spectrum:

- Single programs that run against the NSW file system.
- Single programs that run against their native host file system.
- Related sets of programs that run against either the NSW file system or their native host file systems.
- Entire operating systems.

Two dimensions of interest emerge here. One is the "size" or "extent" of the service. This can range from a single program to an entire operating system. The second dimension is the environment in which the service operates. Here there seem to be two clearly identifiable extremes. One is the native host environment; we will refer to such services as "native" services. The other is the NSW environment; we will refer to these as "friendly" services.

For a friendly service, data access functions, such as file name lookup and file movement operations for user files, would be provided, more or less automatically, as part of the environment within which the service operates. On the other hand, users of native services that access data will, in general, have to use native host file name syntax as well as explicitly initiate file movement operations to make NSW file repository data available to the services.

From a user's point of view all services should be friendly. However, from an implementation point of view it would be very costly, if possible at all, to integrate all services of interest into NSW in this manner. Thus, NSW should support access to, and facilitate use of, both native and friendly services.

Now we consider how NSW might provide environments for services on service bearing hosts. Most hosts support the notion of an active user session, sometimes called a job, which provides an environment where a user can work independently from other users. Generally, to initiate a session a user must be "registered" with the host operating system. Typically, each such user registration has associated with it a portion of the host system file space which is used to provide a context for the user's sessions. The context includes temporary objects used to support a session, as well as more permanent objects used to provide continuity across sessions. Since the exact details of the entity that must be registered vary from host to host and are not important to the present discussion, we shall call such an entity a user slot. Thus, a user slot is a host's long term record of a user and the results of any previous activity, and a user session is a particular activity of the user within the context defined by the corresponding user slot.

From the NSW's point of view, every service, including one running on a user's principal host, runs in a service "workspace". The workspace concept will be developed more carefully in system design documents. At present it is sufficient to say that a workspace provides an execution environment for a service, that workspaces are bound to hosts, and that a user with a principal host will have a permanently assigned workspace on that host. The software that implements NSW will use host user slots to implement the NSW workspace concept. Among other things, this software will transform user slots from different hosts which exhibit different properties into NSW workspaces with more uniform properties. Details of the implementation will, of course, vary from host to host.

One way NSW could provide workspaces for users with principal hosts would be to obtain a collection of user slots from the hosts of interest and to transform them into a pool of workspaces. A workspace from the pool would be assigned to each user that had a principal host. From the point of view of the individual hosts these user slots would be dedicated exclusively to NSW use. They would be owned and managed by NSW, and the only user access to them would be through NSW.

Some users might already have their own slots (i.e., accounts on network hosts) which they would like to have integrated into NSW as NSW workspaces. These users should be able to register these "external" slots with NSW for subsequent use as NSW workspaces. The users would retain "ownership" of these slots. Once a slot was registered, a user could access it through NSW as a workspace, as well as through non-NSW paths as a normal host slot.

The intent is that the properties of a workspace viewed from within NSW would be equivalent whether the workspace was one obtained from the NSW pool or was one supported by a registered "external" user slot.

Access to remote services by any particular user is assumed to be only occasional. Therefore, the level of use of a particular remote service bearing host by a particular user will not justify permanent assignment of a slot for a workspace on the host to the user. To support access to services NSW would obtain user slots on the service hosts and transform them into a workspace pool for dynamic allocation to users as service invocation requires. Slots used to support this pool would be temporarily assigned to users for the duration of a session with a service.

The notion of a principal host can be generalized to allow a user to have multiple principal hosts. In this case the user would have a workspace for each such host. NSW would serve as the access path to the several principal hosts, allowing the user to switch his attention back and forth between sessions on the various hosts. It would also provide for convenient movement of files between the workspaces on the principal hosts, the NSW file system and any active service sessions.

We now turn to a somewhat different kind of user (class U4). These users have no identifiable principal host. For them NSW is the principal system. It provides for their file storage and is their access path to services. Like class U3 users, these users would have need to access both friendly and native services, and for them NSW would provide convenient means to make user data and files available to the services accessed. Users responsible for testing and evaluating new services or software systems fall into this class. The participants in the Ada test and evaluation effort are good examples of these users. Much of the discussion for class U3 users applies to these users; the main exception is that class U4 users would not be assigned host user slots on a long term basis for workspaces.

We consider another type of user (class U5). These are users whose responsibilities are principally managerial. A user in this class might be the manager of a programming project or of some part of a programming project, or the manager of a test and evaluation program. One of their major concerns is controlling access to the resources managed by or accessible through NSW. These users need to be able to grant and revoke other subordinate users rights to access portions of the NSW file repository and various registered services in specified ways. In addition, as personnel join and leave projects, class U5 users need to be able to create and remove users from the NSW system. Decentralization or delegation of responsibility for these actions is highly desirable. By this we mean that once properly authorized, a user should be able to create and remove users without interacting with a central system authority.

IMPLEMENTATION FEASIBILITY

Clearly it is feasible to implement a system that functions simply as a sharing medium or simply as a service access mechanism; the Datacomputer (CCA) and the network access machine (NBS) are examples of such systems. More sophisticated systems that include sharing and access functions, and approach a monitor have also been demonstrated; the best example here is the RSEXEC system (BBN). The user requirements identified above require more than these systems offer.

To be realizable the NSW system concept must, of course, be constrained by what is feasible to implement. We shall try to make the feasibility considerations that have been implicit throughout the discussion above more explicit.

A basic assumption we make is that it is desirable to include more rather than fewer hosts and services in the NSW system. Another is that the development of services is not the primary objective of the NSW project; rather, it is to develop an operating system that can serve as a framework for diverse services, programming environments, and applications.

To participate in NSW a host must be able to perform certain NSW-specific functions. For most this will require software development. The hosts of interest vary greatly in their functional capabilities. For example, some are incapable of supporting friendly services. Hosts can also be expected to vary in the level of their commitment to NSW. That is, regardless of their inherent functional capabilities, hosts are likely to differ in the amount of software they will be willing to develop and run to participate in NSW, and in the amount of local resources they will be willing to commit to NSW use.

This suggests that the NSW system concept must permit, and the system design must define, several levels of host participation. The lower levels of participation would involve fewer NSW specific protocols and require correspondingly less software development. Hosts choosing a lower level of participation would be weakly integrated into NSW and would be likely to support either a limited number of services or only native services. Hosts participating at a higher level would be more strongly integrated and would support friendlier, easier to access services.

System performance considerations must also have an impact on the system concept. Unless the system concept can be supported by an implementation that performs adequately, it will not be used.

The most critical performance factor is interactive responsiveness. Since inter-host interactions can be expected to have relatively high delay, it is important that the system concept not require that such interactions occur frequently during periods of a user session that are highly interactive or that require access only to resources on a single host. It is also true that for some hosts intra-host interactions among software components that implement NSW functions are likely to result in relatively long delays. Both these factors suggest that NSW intervention be minimized during highly interactive periods of a user session, particularly when the user activity requires only local host resources.

SYSTEM CONCEPT

From the foregoing discussion we draw the following conclusions. To satisfy user requirements NSW should be more like a monitor than simply a sharing/communication medium or a service access mechanism. From the point of view of implementation feasibility there are limits to the extent NSW can be a complete, self-contained monitor.

Our concept is for NSW to be monitor-like. It will perform some functions and provide some services monitors generally do. However, since it will function in a cooperative fashion with the monitors in place on the resource bearing hosts, it will differ from single host monitors in several ways:

- It will not perform certain low level monitor functions such as processor management and memory management. There is no need for NSW to do so since the host monitors that are the building blocks for NSW already do so.
- There will be a great variance in the size of the resources it will manage. NSW will provide access to some very large, reasonably complete and self-contained services. While a user operates within the context of these services there will be little need for NSW to intervene to perform its monitor functions. Indeed, due to the way these services are bound to their local host operating systems it may be very difficult to arrange for NSW to intervene.
- To ensure low monitor overhead when a user stays within the context of a single service, NSW will, in certain situations, allow local host monitors to retain responsibility for their normal monitor functions.

The last two points result from the desire to make a large number of services usable through NSW with an acceptable level of performance. The cost for doing so is that a user may, while operating within the context of a "large" service, have to interact with the service by means of conventions specific to the service rather than by means of the uniform NSW conventions.

More specifically, the NSW system will:

- Implement a file system that can serve as a repository for user files and the basis for file sharing among users.

NSW will superimpose standard file naming conventions and standard file operations on top of the disparate filing systems of the underlying hosts. It will be easy for users to move files into and withdraw files from the NSW file system. The file system will provide features enabling users to organize files into groups and to perform operations on groups of files.

- Provide access to a wide variety of services on participating network hosts.

A service might be a single tool, a collection of tools, or an entire operating system. The procedure a user follows to access a service will be uniform regardless of the service and the host that provides it. Only services registered with NSW will be accessible through NSW.

- Provide centralized, uniform access control for objects distributed around the network.

The access control mechanism will be sufficiently powerful to allow controlled sharing of files within the file system, to permit managers to control the access to various registered services that subordinate users have, and to enable the decentralization and delegation of managerial responsibility.

Users of NSW will have user ids and will be required to login to establish their authorization to use the system. Login will also serve to define the access control environment in effect during the user session. The access control mechanism will enable delegation of the permission to create and remove users. It will also enable delegation of the permission to register services.

- Support the notion of principal hosts for users.

Users access to their principal hosts will be through NSW. Once access to the principal host is established, its use should not be degraded by the fact that the access path is through NSW. Although the user's principal host will retain responsibility for user files stored on it, the files will be accessible through the NSW file system. This will facilitate the movement of files between principal hosts and NSW file space. A user may obtain a principal host workspace in either of two ways. First, a user may be assigned one from a pool of workspaces constructed from host user slots previously obtained for exclusive NSW use. Alternatively, a user may already have a slot on some host, in which case he may register it with NSW for use as a workspace. Use of a principal host workspace should be the same regardless of how it was obtained.

- Support access to native as well as friendly services.

A friendly service is one which executes against the NSW file system, whereas a native service executes against its local host file system. The NSW system must provide file movement functions to support both types of services. For friendly services the file movement is activated by the normal operation of the service, and for native services it must be activated by explicit user action.

- Provide support for electronic mail services.

Although not identified above as a user requirement, electronic mail has been established as an indispensable feature. NSW must support it for all the user classes identified above.

- Provide a coherent user interface to the system functions and to the services accessible through it.

The user interface will provide means to invoke system functions such as those of the file system, access control system and so forth. While it will provide uniform access to registered services, the user interface will not attempt to make services appear to behave in a uniform way by filtering or otherwise transforming the interactions between users and services.

- Permit varying levels of host participation.

The level of coherence required by the NSW system concept is infeasible to achieve using only the existing standard network protocols (NCP, TELNET, FTP). To participate in NSW a host must provide certain NSW-specific functions internally and communicate with NSW software on other hosts according to certain protocols. The system design will define levels of host participation in terms of the required function and protocol implementations.

- Support a conceptual model for users that is simple and understandable.

At one time it was felt that a simple conceptual model required host boundaries and the network to be transparent. Experience has shown that transparency of that sort is extremely difficult to achieve, and if not totally achieved can result in user confusion. In our environment complete transparency is infeasible for two reasons: the amount of host software required to achieve it is too great; and providing it, particularly for services as they execute, is likely to result in unacceptable performance.

These realities have led to a system concept that requires some users to deal with more than a single environment. For all users there is the NSW environment. In addition, depending upon a user's requirements, there will be zero, one, or more native host environments. Each environment, of course, has its own language and conventions. The fact that some users may need to interact with the "system" in different languages doesn't necessarily lead to an intractable user model. To be simple and understandable the user model must: embody economy of concept; be predictable and consistent (principle of least astonishment); be probable, in the sense that it should always be possible for users to inquire of the system what the current context and status for their sessions are.

SYSTEM REQUIREMENTS

This section provides more detail about the NSW system concept in five areas: functional requirements, performance requirements, reliability requirements, implementation requirements, and operational and maintenance requirements.

Functional Requirements

- User interface

The user interface will implement a command language that will provide means for users to invoke all NSW functions in a uniform, coherent manner.

- User authentication

User authentication by means of login is required to determine a user's identity in order to establish his authorization to use the system and to establish a context for applying access controls during his NSW session. NSW will implement its own user identification system as the basis for user authentication and access control functions. A login command for user authentication will be provided by the user interface.

- File cataloguing

NSW will provide file cataloguing functions to implement a uniform name space for NSW files regardless of the hosts storing the files. The file catalogue will maintain attributive information about files, including historical attributes. Access through NSW to files on a user's principal host will be supported, although direct responsibility for the files will be retained by the principal host. Operations for grouping NSW files in a "strongly typed" fashion will be provided. This will include means for defining groups, manipulating such a file group as a single object, and manipulating the members of a file group as individual objects.

- File storage

NSW will provide storage space for NSW files.

- File maintenance

Standard file maintenance operations, such as copy, delete, rename and so forth, will be supported for NSW files.

- File movement

Operations for moving files into and out of NSW file space, including movement between NSW file space and user work spaces, will be provided. File movement functions may be initiated by explicit user commands as well as implicitly by the system in support of friendly services.

- File translation

Due to incompatibilities between services and the heterogeneous nature of the underlying hosts, in order to make a file created on one host usable by a service on another it may be necessary to translate the file data or transform the file structure. At a minimum, the system should provide translations between the various "standards" for textual data (e.g., ASCII, EBCDIC) and transformations between sequential and simple record file structures. Information lossless file movement between any two hosts must be possible; that is, it must be possible to move a file from one host to another, possibly to store the file, and back without the loss of any file data or file structure information.

- Service activation

The user interface will provide a uniform access path to services on service bearing hosts and, for users with principal hosts, an access path to the principal hosts. Generally, service activation will be accomplished in a way that will require no explicit user participation in access control check exchanges. Access to batch services, as well as interactive services, will be supported. An interactive means of submitting "jobs" for batch services will be provided.

- Execution environment for friendly services

While not required of all services, some services will be friendly in the sense that users will not have to explicitly initiate file movement and translation operations to ensure that data required by the service is accessible to it. Any file movement and translation involved will be initiated by NSW as required to support execution of the service.

- Registration function

This is a cataloguing function similar to that provided for files. The registration function is the means for integrating services into NSW name space and for introducing "external" host user slots into NSW for use as workspaces.

- Access control

An access control mechanism will provide means to selectively control user access to NSW files, registered resources, and certain NSW functions. The access control mechanism will permit the delegation of managerial responsibilities in a hierarchical fashion. Means to grant and to revoke user permissions will be provided. The ability to create and remove NSW users will be controlled by the access control mechanism.

- Accounting

The system will account for the use of file storage space and host CPU time. Accounting data will be kept on line for analysis by management tools.

Performance Requirements

The functional requirements above must be translated into system functions that are implementable on conventional computer systems and that can be supported by an optimizable system architecture. Otherwise few people will use the system.

Because a large part of the system concept is based on accessing existing software services, it is possible to identify performance criteria for NSW that are based on cost and benefit measures of the following types:

- Sample Cost Measure. A user session with a tool or service within NSW should be no more than X% worse than the same session with the tool in its native host environment.
- Sample Benefit Measure. Use of a set of distributed services within NSW should be at least Y% better than use of the set on their respective native hosts with no network operating system support.

Measures such as these appear to be objective. However, to be useful they need to be made more precise (i.e., "better" or "worse" over what domain?, real time delay, number of user key strokes, amount of user knowledge required?, etc.), and then to be transformed into performance objectives (i.e., for each domain, what are the required X and Y?). Furthermore, such criteria are difficult to test since test results (i.e., measured Xs and Ys) are likely to depend strongly upon the details of the benchmark scenarios and the experience of the users.

Rather than set performance requirements of this sort for NSW, we make the following observations. It is reasonable to expect a cost to be associated with the benefits provided by NSW. The cost is due to the monitor overhead required to achieve the benefits, and its most visible manifestations are likely to be slower interactive responsiveness and a corresponding reduction in the number of simultaneous active user sessions the hosts that are part of NSW can support at a given level of performance. The system concept described above recognizes the cost issue and attempts to address it by incorporating means to reduce the cost impact while maintaining the system benefits. For example, the notions of principal hosts, native services, and permanent workspaces are all intended to minimize NSW monitor involvement. Furthermore, the system concept recognizes that not all users will require the same level of service. Users who choose not to use features likely to require significant monitor overhead (e.g., friendly services) should not incur significant overhead due to them in the NSW features they do choose to use.

In the final analysis, the users will judge if the system performs adequately for the benefits it provides them. This judgement will be largely subjective.

Reliability Requirements

As with performance, NSW must provide an adequate level of reliability or it will have no users. We identify four areas of concern that must be addressed by the system:

- System availability

To a certain extent the availability of the system depends upon the availability of the underlying hosts and communication network. These factors are, of course, out of our control as NSW designers and implementers. The fact that the system concept permits varying levels of host commitment to NSW only makes the problem more difficult.

To approach this problem, we identify a set of core functions (e.g., user authentication, access control, file and service cataloging) without which the system cannot be regarded as operational. Other functions, features and services (e.g., specific user files, specific services) are less critical in the sense that they can be unavailable without impacting the ability of the system to perform the core functions. We will call the implementation of the core functions the core system, and require that it be "highly" available. "Highly" must, of course, be measured against the availability of the core system hosts. It is clear that the core system should be implemented on hosts that are highly available and that have a high commitment to NSW.

- Failsoft operation

The system should be able to continue to function in the presence of communication or host outages, perhaps with reduced functionality or with degraded performance. The division of the system into core and non-core parts, in part, addresses this issue. The core system should be able to operate in the presence of failures in the non-core system parts. The loss here, of course, would be the functions provided by the non-core parts. Furthermore, it is feasible to replicate the implementation of certain core functions, such as user authentication, to make them less vulnerable to failures of core system hosts. When a user is operating entirely within a service, failures of any other parts of the system, core or non-core, should not interfere with his service session; that is, the non-availability of system functions not required should not affect the user.

- Recovery of partial results

When a user session is interrupted by a component outage, it should be possible at a later time to recover work partially completed during the session. Specifically, files which are "trapped" in a workspace due to a failure should be retrievable when the failed components are re-integrated into the system. Recognizing that the extent to which NSW can satisfy this requirement depends upon the constituent hosts, we relax it somewhat and require that the ability of NSW to recover partial results in a given situation should be no worse than that provided by the host(s) in question.

- Integrity of stored data

It is important that data accepted for storage by NSW be protected against loss or corruption due to host crashes. Specifically, NSW should not lose user files. Again, this requirement must be moderated somewhat to recognize that not all hosts provide the same level of reliability in this area and no host is completely invulnerable to catastrophic events (e.g., disk crashes, fire in the computer room).

Implementation Requirements

The NSW implementation should:

- Support varying degrees of host commitment.

The system architecture should be structured so that each host computer can determine the extent of its commitment to NSW. There should be minimal requirements for a host to participate in the system, and a range of requirements for more sophisticated host involvement. There should be well defined "levels" of implementation so that a host can make choices in terms of the NSW functionality it will implement, the performance optimizations it will provide, and the reliability and failure recovery mechanisms its NSW software will incorporate.

- Support customized user access points.

The motivation here is for a system architecture that can accommodate user access points that take advantage of new technologies (such as multiple window displays) as they are developed, as well as accommodate limited capability user access points (such as ARPANET TIPS). Effective support for new technology user access points is likely to require the ability to provide a reasonably direct path between user terminals and the services they are using. By "reasonably direct" we mean that the bandwidth and throughput of the path should not be significantly reduced by NSW overhead.

Operation and Maintenance Requirements

Operation and maintenance procedures for NSW can be expected to be more complex than those for single host operating systems. However, the system can provide means for dealing with the complexity. For example:

- It should be possible for operators to perform routine operator functions through the system itself as logged in users.
- The system should implement a set of status functions sufficient to enable an operator to determine the status of the system as a whole and to obtain detailed information on the state of various system components.
- The core system should provide a uniform logging capability to be used by system components to record errors, faults and exceptional conditions as they are detected.
- It should be possible for an operator to start the core system from a single point (terminal).
- There should be software that can be run (offline) against critical system data bases to verify their integrity and, if necessary, to restore them to a consistent state.
- It should be possible for an operator to take the system down in an orderly fashion from a single point.
- It should be possible for an operator to isolate components suspected to be malfunctioning from the core system, and later re-integrate them when they are judged to be properly operating.
- There should be well defined procedures for interacting with hosts which are in the non-core parts of the system, and it should be possible for an operator to start and stop parts of the non-core system from a single access point.

NSW Functional Specification

Prepared by
Bolt Beranek and Newman Inc.

D R A F T

Original Draft: June 1980
Revised: September 1980

Appendix B

Table of Contents

1. Introduction	1
2. Relation to NSW 4.1	3
3. Major System Elements	6
4. Overview of Major System Functions	9
5. The resource catalogue model for retained objects	16
6. Retained Objects: Files	21
6.1 Basic File System Design	22
6.2 Set Files	29
6.3 File Specs, Scopes, Ellipsis, and the Lookup and Entry Functions	34
7. Retained Objects: Devices	40
8. Retained Objects: Services	44
9. Retained Objects: Workspaces	57
10. Nodes and Permissions	66
10.1 Rules for Permissions	66
10.2 Nodes and Permissions for NSW	67
10.3 Node Names	74
10.4 Own Space	76
10.5 Permission Types	78
10.6 Node Records	80
10.7 Granting Permissions	82
10.8 Summary	83
11. User Sessions	84
12. Other Aspects of the Design	86
12.1 Status Functions	86
12.2 File Placement	90
12.3 Importing and Exporting Files	90
12.4 File structure, representation, translation	91
12.4.1 Representational Issues	91
12.4.2 File Movement from Host to Host	91
12.4.3 File Translations	92
12.5 File Semaphores	93
12.6 File Version Numbers	95
12.7 File Attributes and Tags	98
12.8 Additional Levels of file storage	100
12.9 Support for Configuration Control	100

12.10 Mail	101
12.11 User-User Interactions	105
12.12 Information services	106
13. Operator Functions	107
14. Summary of User Model	108
14.1 The NSW File System	108
14.2 Conversational Partners	111
14.3 Types of Services	112
14.4 Summary of Naming Conventions	113
15. System Software Architecture	115
15.1 Major Functional Modules	115
15.2 User Access Points	121
15.3 Intercomponent Protocols	126
16. Implementation of NSW Host Supporting Software	131
I. APPENDIX: On the Implementation of Workspaces from Host User Slots	133

List of Figures

Figure 5-1:	17
Figure 6-1:	26
Figure 6-2: A native host set file.	30
Figure 6-3: An NSW set file.	31
Figure 6-4: An open NSW set file.	31
Figure 6-5: A closed NSW set file.	32
Figure 8-1: Conversational partners for a program service.	47
Figure 8-2: Conversational partners for an ICP service.	48
Figure 9-1:	64
Figure 10-1: Relation between exist and cd permissions.	68
Figure 10-2:	73
Figure 10-3:	83
Figure 12-1:	95
Figure 15-1:	117
Figure 15-2:	121
Figure 15-3:	122
Figure 15-4:	123
Figure 15-5:	124
Figure 15-6:	125
Figure 15-7:	126

List of Tables

Table 8-1:	50
Table 16-1:	134
Table 16-2:	134

1. Introduction

This report presents a functional specification of the NSW system. It is the second in a series of documents that specify the NSW system. The first document in the series was titled "NSW-CONCEPT", November 14, 1979. It identified system requirements for NSW and developed a system concept that addressed the identified requirements.

The system functional specification is presented by developing a conceptual model for the NSW system in terms of the system functions and the underlying system software architecture that implements the functions.

The report describes a system, an important aspect of which is the interrelationships of its parts. Because the presentation is sequential, some parts of the system must be explained before other parts. We attempt to minimize the difficulty this might pose to a reader by providing an overview of the system before describing the various parts in detail. Sections 3, 4 and 5 provide overview information.

The purpose of this report is to describe the nature of the NSW system, not to specify all of its details. Thus, many details are omitted from this report, and are left for more detailed design documents. In particular, considerable implementational detail is omitted.

At present an implementation of NSW, NSW version 4.1, is operational. The relation of that implementation to the design specified here requires explanation. In a sense NSW 4.1 represents a partial implementation of the design specified in this report. That is not surprising since the design effort that led to this report was motivated by performance and functional limitations of NSW 4.1. The design effort was constrained by a desire to preserve as much as possible of the software investment represented by NSW 4.1. Consequently, a number of approaches which might have been investigated were not. For example, the possibility of distributing the data bases used to catalogue and control access to various NSW objects, such as files, was not seriously considered since a significant part of NSW 4.1 is the software that supports a centralized collection of catalogue and access control data bases.

Despite these design constraints, the design presented here represents a substantial improvement to NSW. The design is such that the ultimate NSW system it specifies can be achieved, starting from NSW 4.1., through a sequence of system releases, each of which provides increased functional capability.

Section 2 discusses how NSW 4.1 differs from the design in this report. Another document, titled "Enhancing the NSW System",

addresses the issue of how NSW 4.1 could evolve into an implementation of this design.

2. Relation to NSW 4.1

As noted in Section 1, there is an operational implementation of NSW, called NSW version 4.1. NSW 4.1 implements some of the design documented in this report. In fact, experience with NSW 4.1 and earlier NSW implementations motivated much of the design work described here. This design work began with an analysis of NSW 4.1.

In a sense this "new" design represents enhancements to NSW 4.1 intended to improve its functional capabilities and its performance properties. In addition, it differs from NSW 4.1 by permitting varying degrees of host participation in NSW by defining levels of host implementation of NSW support functions, and by supporting a wide variety of user access points.

We believe that it is feasible for NSW 4.1 to evolve in an orderly sequence of steps toward the design presented in this report. Each step would be realized by a version of the system that provides functional and performance improvements over the previous step, and would in its own right represent a useful NSW system.

We conclude this section with a list of some significant differences between NSW 4.1 and the new design. Familiarity with NSW 4.1 is, of course, assumed. The items on the following list cite sections of the report that discuss them in detail.

1. The notion of an NSW Resource Catalogue is fully developed in the sense that all objects manipulated by users through the system (files, tools, nodes, etc.) are catalogued in it. (Sections 3, 5, 6, 7, 8, 9.)
2. Printer devices are accessible through NSW, enabling users to produce listings of printable NSW files. This is accomplished, in part, by cataloguing devices and the procedures required to access them in the NSW Resource Catalogue. (Section 7.)
3. Devices can be introduced to NSW by means of user level commands which enter the devices into NSW Resource Space. (Section 7.)
4. Electronic mail services are supported by the system. (Section 12.10.)
5. Catalogued objects, such as files, may have multiple names. (Section 6.)
6. The notion of scopes is generalized. (Sections 6.3, 7, 11.)

- o Scopes apply to all objects (e.g., devices and services as well as files) recorded in the Resource Catalogue.
 - o A user can instruct the system to use scopes in a series fashion rather than in a parallel fashion if he so chooses.
 - o Separate commands are provided for changing the scopes that are to be in effect for the current session and for changing a user's default scopes which are stored in his node record.
7. The system supports compound files called set files. A set file is made up of a collection of members each of which is itself a file. There are two types of set files: NSW set files which together with their member files are managed by NSW core system software, and native host set files whose management requires supporting software on the hosts that store them. (Section 6.2.)
8. The notion of tools has been generalized and renamed to service. (Section 8.)
- o Services are objects which are catalogued in the NSW Resource Catalogue.
 - o NSW 4.1 interactive tools are a type of service called program services.
 - o NSW 4.1 batch tools are a type of service called batch services.
 - o A new type of service called a workspace command interpreter is introduced.
 - o A new type of service called an ICP service is introduced.
 - o Program services can run in native file access mode as well as NSW file access mode.
9. Services can be introduced to NSW by means of user level commands which enter the services into NSW Resource Space. (Section 8.)
10. Workspaces are elevated to first class NSW objects and may be catalogued in the NSW Resource Catalogue. (Section 9.)
- o In their ability to store files, workspaces

exhibit properties similar to those of native host set files.

- o There are user level commands for moving files between NSW file space and workspaces.
 - o Workspaces can be allocated on a long term basis (i.e., across sessions, for months) to users.
11. Users can register external host accounts with NSW for use as NSW workspaces. The result of such registration is to enter the external account into NSW name space as a workspace object. (Section 9.)
 12. A permission system for NSW is fully developed. (Section 10.)
 13. User nodes are treated as NSW objects. (Section 10.)
 14. The notion of "own" space, which is a region of NSW resource space to which a user has exclusive access, is supported. (Section 10.)
 15. There are user level commands for instructing NSW to move NSW files from one NSW host to another NSW host. (Section 12.2.)
 16. Version numbers are supported for NSW files. (Section 12.6.)
 17. Features for real time user-to-user interactions, such as operator-to-user messages, and user-to-operator and user-to-user dialogues, are introduced. (Section 12.11.)
 18. Users can import files from and export files to ARPANET hosts that don't implement NSW file packages. (Section 12.3.)
 19. A wide range of user level status commands are supported. (Section 12.1.)
 20. A means of supporting user access to NSW from a variety of network access points, along the lines of the "parser/switcher" model, is supported. (Section 15.2.)

3. Major System Elements

This section briefly describes the principal parts of the NSW system.

The NSW is a network operating system. In very general terms, the NSW system maintains objects for users, and it provides means by which users may access these "retained objects". The retained objects it manages reside on different host computers that are connected to the ARPANET.

Examples of retained objects managed by NSW are files, devices and services. Each of these objects and others are described in detail in this report.

The NSW uses several data bases to provide its services to users. These data bases include the following:

- o NSW Resource Catalogue

There is a record, called a catalogue entry, in the resource catalogue for each retained object. The information in the catalogue entry includes the type of the object as well as information required by the system to provide user access to the object.

- o Session Data Base

There is a record in the session data base for each active NSW user session. The active session record includes the identity of the user and other information required by the system to manage the user's active session.

- o Node Permission Data Base

There is a record, called a node, in this data base for each user of the system. The node includes information about the user such as his login name and password, a set of "permissions" which define the actions he has been authorized to perform within the system, and the specification of default settings for various parameters governing his use of the system. Information in the node is used to initialize various fields in the user's active session record when he logs in.

In general terms the NSW software does the following things:

- o It modifies the state of retained objects as the result of user actions.
- o It modifies the state of a user's record in the Session

Data Base as result of user actions.

- o It provides access to retained objects (i.e.,resources) by establishing access paths to the resources.

The software that implements the NSW system can be partitioned into the following elements:

- o NSW Core Software

This software makes direct use of the data bases introduced above to perform functions such as user authentication, user session management, access control, and resource management required to satisfy user requests. The last function includes finding objects given their names, and establishing access paths to them.

- o Host NSW Supporting Software

This software integrates a host into the NSW system. It provides access to the resources and services on its host that are to be reachable through NSW. The functions it implements include program activation and control, file maintenance, and file movement into and out of its host. Initiation of these functions is usually at the request of NSW core software.

- o User Access Point Software

This software provides the user's interface to the NSW system. It implements a command interpreter that accepts commands from the user and interacts as necessary with the other system software to accomplish the action requested.

- o Host Operating System Software

NSW is designed to be built upon the operating systems of the host computers that are part of it. Thus, the operating systems of the constituent hosts are, in a sense, part of the NSW system.

- o Host Application Software

One of the things NSW does is provide access to services on various hosts. Many of these services are implemented by application level (as opposed to operating system level) software on the hosts. Thus, host application software is also, in a sense, part of the NSW system.

This report is concerned primarily with the first three of these software elements.

Another aspect of the system is a set of "intercomponent protocols". These protocols are the conventions used by the various software elements to communicate with one another. For example, the interactions between the NSW core software and host NSW software are governed by these protocols. The protocols are structured into a number of layers. Different protocol layers specify the patterns of interactions among the software elements to accomplish various system functions, the content and format of the interactions, and the means by which interactions are to be accomplished between software elements on different hosts.

An important design principle for NSW is that any system function that can be invoked through the user access point by command can also be invoked by a program initiated call on the system function.

4. Overview of Major System Functions

This section introduces most of the major system functions. It does this by briefly describing their semantics in an intuitive fashion in terms of their use of and effect upon the data bases introduced in the previous section.

Each of the function descriptions includes a command a user might initiate through a command interpreter. The function invoked by the command is described in terms of what the system software does to perform the function.

We defer for later a discussion of how a user might reach a command interpreter through a user access point (See Section 15.2).

A stylized command language, not unlike that implemented for NSW 4.1, is used for specifying the commands in the function descriptions. For the most part, we shall rely upon the reader's intuition regarding the command language.

The function descriptions follow:

1. Login

The login function is the means by which a user session is initiated. Typically a user logs in to NSW by means of a login command:

```
NSW: login schantz zxcvbn
      Richard E Schantz logged in
```

Here "schantz" is the user's login string and "zxcvbn"
1
is the user's login password¹.

The system uses the login string and password to retrieve the user's node record from the Node Permission Data Base and verify the password.

If the specified password is correct, a new active user session is created. This is accomplished by adding a new active session record for the user to the Session Data Base. Various fields in the new active session record are initialized from information in the user's

1

When the user types the password, printing would normally be suppressed. It is shown here for illustrative purposes.

node. These include the user's id (i.e., the node id) and the permissions in effect for the user.

2. File Manipulation

Files are an important type of retained object managed by NSW. NSW implements a network file system using the file systems of the constituent hosts. The NSW file system has uniform file naming conventions which are different from those of the constituent host operating systems. The constituent host file systems are used principally to provide storage containers for NSW files. The NSW file name space is implemented by the NSW Resource Catalogue.

The system provides means for users to manipulate and maintain files. The following illustrates deletion of an NSW file.

NSW: delete (file) a.b

Here the user has instructed the system to delete the file known to him as "a.b".

To carry out this command the system performs a lookup function consulting the Resource Catalogue to find the catalogue entry corresponding to the named file. The catalogue entry contains descriptive information about the file including the host (H) that stores the file and the name (HF) that the file is stored under at the host.

Prior to acting on the delete request, the NSW core software checks the Node Permission Data Base to verify that the user is allowed to delete the file. To do this it checks whether the user's node holds a permission that authorizes the user to delete the file.

To delete the file, NSW core software instructs NSW supporting software on host H that stores the file to delete the file HF, and then removes the catalogue entry for the file from the NSW Resource Catalogue.

As noted above, the Resource Catalogue defines a name space for files and other objects retained by NSW. To allow users freedom in their choice of file names and to avoid naming conflicts the system supports the notion of regions of the name space, and file names such as "a.b" are generally interpreted within the context of some subregion of the entire space. The particular subregion(s) in effect for a user is part of the user's session context, and is defined by one or

more "file scopes" which are maintained in the user's record in the Session Data Base. The scopes initially in effect after a user logs in are set from scopes stored with the user's node in the Node Permission Data Base.

3. Run a Program

Services are another type of object managed by NSW, and program services are an important type of service. The following illustrates starting a program service known to the user as "teco".

NSW: use teco

Instance name is teco; workspace is Thomas1.r20

Now talking to teco; using NSW file access mode

TECO 2.35.4 10 Oct 79

*

.

.

.

^N

NSW:

To start the program the NSW software first consults the Resource Catalogue to find an entry for the named program. Descriptive information about the program such as the host that runs it and other information required to properly start it are stored in its catalogue entry.

The NSW core software uses this information to start an instance of the program for the user. It does this by interacting with NSW supporting software on the program host, instructing the host to establish a communication path between the program and the user to support interactions between them.

After the program has been started and the communication path established, the user's session record in the Session Data Base is updated to reflect the fact that the user has a new program instance.

At this point the user has two conversational partners, the NSW command interpreter and the interactive program, and he may switch his attention back and forth between them by means of commands to the command interpreter. The program instance may be referred to in these commands and others by its "instance name" which in this example is "teco".

A program service operates within a "workspace" on the

host that supports the program. The workspace defines an execution environment for the program, in part by providing space for holding files accessed or created by the program. Workspaces are an object type maintained by NSW. Like other objects, workspaces may have names and the system supports a set of operations for them (see Section 9). In this example the workspace for the teco program is named "Thomas1.r20".

Many programs reference files specified by users. In this example the program treats names of files specified by users as names of files in the NSW file system ("NSW file access mode"). Other modes of file access are also supported (See Section 8).

4. List a file

Devices, such as printers, are objects managed by NSW. Like other objects, devices are part of NSW resource space, they have names, and they are catalogued in the Resource Catalogue.

In the following example, the user instructs the system to make a listing of a file.

NSW: print MSG.Aspec

Part of a user's context is a set of standard devices including a "session terminal" and a "line printer". The correspondence between the user's printer and an actual line printer device is maintained with the user's session record in the Session Data Base. This correspondence is initialized from information stored in the user's node and there are commands for a user to change the correspondence, either in the session record or in the node record.

To carry out this print command the system must locate the file named "MSG.Aspec", locate the line printer that is part of the user's context, and "move" the file to the printer for printing.

The file is located, as in the example above, by consulting the Resource Catalogue using the file scope(s) in effect and the file name provided by the user. An access control check is made to ensure that the user's node holds a permission authorizing him to print the file prior to printing the file. The line printer in effect for the user is obtained from the Session Data Base. To cause the file to be printed, NSW core software interacts with NSW supporting

software as required to move it to the printer (see Section 7 for more details).

5. Create a new user

Users may add new users to the system by creating new user nodes in the Node Permission Data Base. To create a new node the node name, a login string and other information about the user must be specified as the following illustrates.

```
NSW: create node
      node type: person
      Person's last name: Thomas
            first name/initial: Robert
            middle name/initial: H
      Person's login string: Thomas
      Person's password will be asdfgh
      Own space name will be: $Thomas1*
      ... other information about new user ...
      Person node
        Thomas.Robert.H.1
      created
```

Like most actions supported by the system, creation of a new node requires a permission. Thus, prior to actually creating the node, the system first checks the user's own node to ensure that it holds a permission authorizing the user to create nodes.

The right to be known as a user of the system is governed by the permission system in the sense that there is a type of permission known as an "exist" permission which must be held by a node in order for the node to have a record in the Node Permission Data Base. In these terms, part of node creation is granting an exist permission to the node, and part of node deletion is removing a node's exist permission.

The system creates a node by adding a new node record to the Node Permission Data Base, giving the new node an exist permission, and storing other node information in the new node record. As soon as the new node record is added to the Node Permission Data Base the new user can log into the system.

Like other system objects, nodes have names. The node

created in this example is named "Thomas.Robert.H.1"². Users are granted their own regions of NSW resource space. In this example, the new user has been assigned an "own space" region named "\$Thomas1*".

Nodes, permissions, names, and own spaces are described in detail in Sections 10 and 14.4.

6. Send a Message

NSW supports electronic mail services which enable users to send messages to one another. Mail for a user is delivered to the user's "mail box" from which the user may read it. From a user's point of view mail services are accessible by means of a mail program as the following illustrates.

```
NSW: mail
      now talking to mail
```

```
Mail System 3.2.5 17-Oct-79
>compose message
To: Thomas
Cc: Schantz
Subject: NSW protocols
Text:
The NSW protocols ...
^Z
Send? yes
Message sent
>
```

Here the user has composed and sent a message.

To carry out this scenario the system starts up an instance of the mail program much as described above for the teco example. After the mail program has been started and a communication path to the user established, the user may deal with the mail program as a conversational partner.

For this example, the system must locate mail boxes for two users, Thomas and Schantz. The location of a

2

This is the full node name. Users generally use "node specifications" rather than full node names to refer to the node. For example, the string "Thomas" might be a node specification for the new node.

user's mail box is part of the information stored in the user's node record in the Node Permission Data Base. The system consults the Node Permission Data

3

Base for nodes for users Thomas and Schantz . From the node records for these nodes it can obtain the locations of their mail boxes, and then call upon NSW supporting software as necessary to deliver the messages. Section 12.10 describes mail services in more detail.

3

To do this it makes use of a lookup function for the Node Permission Data Base that accepts the strings "Thomas" and "Schantz" as node specifications.

5. The resource catalogue model for retained objects

As the examples in Section 4 suggest, 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 objects and agents. Subsequent sections describe each of the object types in detail.

Objects are the elements which make up NSW resource space. Examples of objects are files, devices, and services. As mentioned in Section 3, NSW resource space is defined by the Resource Catalogue. The Resource Catalogue and the NSW software that uses it, in effect, implements a name space for objects. The terms "NSW resource space" and "NSW name space" are used synonymously in this paper.

Nodes are the system's representation of the agents (i.e., the users) that manipulate NSW objects. Nodes for users are maintained as records in the Node Permission Data Base. Permissions are associated with nodes, and specify what actions the agents represented by the nodes may take with the objects. Typically, a permission identifies an object or a region in NSW resource space and the operations the node that holds the permission may perform on the object or region. The nodes that represent agents are themselves objects. That is, from within the NSW system nodes may be created and deleted, and permissions may be granted and revoked.

Some types of NSW objects are built more or less directly out of similar objects provided by the operating systems of the constituent hosts. Examples of these objects are files, workspaces and services. Other kinds of objects are realized entirely by NSW. Examples are nodes and permissions.

Every object that can be manipulated through NSW has an NSW name. The name-to-object correspondence is maintained by the

4

Resource Catalogue .

A number of operations are common to all object types. These

4

We shall see shortly that there is not necessarily a single data base which is the Resource Catalogue. For example, nodes are objects but are maintained in the Node Permission Data Base. Nonetheless, from a conceptual point of view it is useful to think in terms of a Resource Catalogue, and we shall continue to do so with the understanding that the Resource Catalogue may, in fact, be realized by a number of different data bases.

operations are:

1. enter

This function establishes a binding between a name and an object. It does this by creating a new catalogue entry that records the correspondence between the name and the object.

2. delete or remove

This breaks a name-to-object binding by removing the catalogue entry for the object. Depending upon the object type, it may have additional semantics.

3. lookup

Given a name, this function finds the object (or more precisely, the catalogue entry for the object) corresponding to the name.

4. rename

This function binds a new name to an object, thus changing the name-to-object binding for the object.

Figure 5-1 illustrates the relationship between names, catalogues and objects.

name <---->	catalogue ---->	i-node ---->	object
of	entry for	(maintained	
object	name	by NSW; holds	
	(a record in	object descriptor	
	the Resource	that includes:	
	Catalogue)	host, host name for object	
		...)	

Figure 5-1:

The i-node (index node) contains a descriptor for the object which holds all of the information required to initiate operations supported by the system for the object. Thus, the i-node, in a sense, is the catalogued object. The i-node is separated from the catalogue entry to allow for the possibility that an object might have multiple names.

As noted above, before an object can be manipulated by the

system, it must be "entered" into NSW resource space. Conceptually entry involves several separate steps. If the object does not yet exist, it must be created. Next, a descriptor for the object is created. Finally, a new catalogue entry for the object is added to the Resource Catalogue which establish the various linkages that relate the object's NSW name, its i-node, and the object itself. For example, consider the steps involved in entering a new file into NSW resource space:

1. The file object is created. This is accomplished on the host that stores the file and is not directly an
5
NSW operation. The result of the file creation is the file and a "handle" for the file that NSW can use, called a "file id". A file id identifies the host that stores the file and the name the file is known by on the host.
2. An object descriptor for the file is created. The object descriptor includes the file id as well as other descriptive information about the file such as the date and time of its creation, its size, etc.
3. A catalogue entry for the file is added to the Resource Catalogue. The catalogue entry includes the NSW name for the file and identifies an i-node that holds the object descriptor for the file.

There are NSW conventions for naming objects in NSW resource space, and for naming regions of the resource space.

The complete NSW name for an object is a sequence of name components. It is sometimes useful to think of an object's name as the name for a path through NSW name space to the object. Syntactically we write:

a.b.c.d.e

for the object with name components a, b, c, d, and e.

Users seldom specify complete names when referring to objects. Generally, they specify incomplete or partial names which the system interprets according to its name lookup rule (See Section 6.3). For example, a user might use

5

That is, the NSW does not directly perform the disk operations, or the open or close operations required to create the file object.

d.e

to refer to the object whose complete name was a.b.c.d.e. To accomplish the lookup the system may make use of additional information, such as that found in the user's session record (for example file scopes). To avoid confusing the names of objects (which are fully specified sequences of component names) with the specifications users supply for objects (which usually only partially specify the component names) NSW uses the term "object specification" (or "spec" for short) to refer to user supplied object specifiers.

A user may want to fully specify the name components for an object in a spec. To do so he must signal the system that the spec includes all the name components in order to disable the function that transforms a partial spec to a complete object name. The symbol "\$" is used for this purpose. For example, the user supplied spec

\$a.b.c.d.e

refers to the object whose complete name is a.b.c.e.e.

When interacting with NSW, users always refer to objects by means of specs. Similarly, when the system must refer to an object in interacting with a user, it always prints a spec for the object rather than its name. Thus, to refer to the object named a.b.c.d.e, the system would print the spec

\$a.b.c.d.e

whenever it is necessary to specify all of the name components; if a partial spec was appropriate, it could print the spec

d.e

To repeat, at the user interface, objects are referenced by specs rather than by names, both by users and by the system.

A region of NSW name space is named by a sequence of name components. It designates the region of name space that includes objects whose names begin with the sequence of name components. For example,

a.b.c

names a region of NSW resource space that includes every object whose name begins a.b.c.

At the user interface the "\$" convention is used in region specifications. In addition, to emphasize that a region is being specified, the symbol "*" is used following the sequence of name

components. Thus, at the user interface the region named a.b.c is specified by

\$a.b.c*

Region specs are used in "scopes" (see Sections 4 and 6) to localize the domain of the lookup function, and in various types of permissions (see Section 10) to define the objects governed by the permissions in terms of the region of name space they occupy.

6. Retained Objects: Files

The file system design is strongly influenced by two considerations:

1. It is desirable to allow a many to one correspondence between file names and files.

We recognize that this represents a significant departure from the current NSW file system design. However, experience with modern operating systems such as Multics, Unix, and TOPS-20 has shown this notion, typically implemented as a file name synonym feature or a file link feature, to be a very useful one for supporting flexible file sharing among users. Some of the configuration control mechanisms suggested for the maintenance of NSW itself (e.g., "package logs") seem to require such flexible file naming and sharing, and user level implementations of these features are likely to be developed if the system does not provide adequate support for them.

The particular mechanism we incorporate in our design⁶ is a file linking mechanism .

2. It is desirable to provide support for groups or collections of files in a much more "strongly typed" fashion than merely collecting together file names.

This support should include means for manipulating such a file group as a single object as well as for manipulating the members of a file group as individual objects. We shall call such a group a "set file". The set file notion addresses requirements in a number of areas. These areas include software configuration control which deals with "packages" of files that taken together represent a version of a software module, tool naming and tool management, and libraries and partitioned data sets found on some tool bearing hosts.

The next section presents the basic file system design. It focuses on file naming and access control. The set file notion

6

The ability to have multiple names for a single object may not be implemented in the immediate future. However, we document the design here to ensure that the feature can be easily accommodated by the implementation.

is presented in Section 6.2. Section 6.3 then discusses some further aspects of file naming.

6.1 Basic File System Design

A complete NSW file name is a sequence of name components. It is sometimes useful to think of a file name as a path name through NSW file name space. Syntactically we write:

\$a.b.c.d.e.f

to specify the file with name components a, b, c, d, e, and f.

The file system's record of a file name is a "catalogue entry" (CE). There is a one-to-one correspondence between file names and CEs. A CE contains sufficient information to allow the system to access a particular NSW file. However, it is important to note that a CE is distinct from the file it catalogues.

Part of the effect of the lookup function (see Section 5) for files is to find the CE corresponding to a given file spec. Part of the effect of the entry function is to establish a new file name-to-CE correspondence. Similarly, part of the effect of the removal function is to break the correspondence between a file name and a CE.

To enable the NSW permission system (described in Section 10) to control user access to files, a set of rights that govern file access must be defined. Rights could conceivably be on a per file basis; that is, to access a file a user could be required hold a permission for that file. There are two factors that make 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 if any sharing is permitted.

Clearly an approach which employs rights for groups of files is called for.

The approach in the current NSW system avoids these two problems nicely. A file permission is for a region of NSW file space. Such a permission is called a "key". A key specifies a

region of NSW name space and an allowed access type.
Syntactically we write:

[T, \$a.b.c*]

for a key for the region of file space designated by \$a.b.c* that allows access of type T.

The access control check function is conceptually quite simple. When a user attempts to access a file in a particular way, the system checks the user's node for a permission with a key that is of the appropriate access type and is an initial substring of the file's name.

Two minor difficulties result from the fact that a key defines a region of file space:

1. It is difficult to grant a user access to some but not all files in a region. If this type of control is required, in place of a single key, several more restrictive keys could be used to define the files for
7
which access is permitted .
2. A key may permit access to as yet non-existent files. For example, one user might grant another access to the file \$a.b.c by giving the key \$a.b.c* to the second user's node. Should the first user later create files named \$a.b.c.d and \$a.b.c.e, the key would allow the second user to access them. Some users may find that this property places annoying restrictions on their use of file name space and keys.

To help overcome these difficulties NSW will support the notion of keys for single files in addition to keys for regions of file space. Thus, the key

[T, \$a.b.c]

7
Another approach would be to introduce the notion of "prohibitions" which are opposites of permissions. A prohibition would prevent rather than allow actions, but would follow rules similar to those to be developed for permissions in Section 10. A permission key could then be used to grant access to an entire region of file space subject to restrictions specified any prohibition keys (perhaps "locks" is a better term) held by a node. We don't develop this notion further in this note.

will apply only to the file \$a.b.c and not to the files \$a.b.c.d or \$a.b.c.e.

At this point we identify two types of file access:

- o enter.

An enter key permits the enter function to be used for the portion of NSW file name space designated by it.

- o delete.

A delete key enables use of the removal function for the portion of name space designated by the key.

Later other access types such as copy and execute, and the
8
corresponding keys, will be introduced .

A user's keys are recorded with the user's other permissions in the Node Permission Data Base (See Sections 3, 4 and 10). In addition to these "private" keys the system maintains a table recording "public" keys which system administrators have determined should be available to all users. When an access control check is made, a user's rights are defined by the union of his own keys with any public keys. Public keys are a simple mechanism to avoid replicating keys in every node and to support easy updating of keys common to all users.

The same NSW object may have more than one NSW name. Thus, there may be a many-to-one correspondence between file names and files, and as a result there may be many CEs which catalogue a particular file. This notion is incorporated into the system by a feature called "linking".

Recall from Section 5 that the system's record of a object is an index node. There is a one-to-one correspondence between i-nodes and objects. A CE identifies an i-node, and many CEs may identify the same i-node.

A more complete description of the lookup, entry, removal, and

8

It might be useful to introduce the notion of "lookup" access and "lookup" keys. A lookup key would enable use of the lookup function for the designated portion of the name space. The lookup function would fail if the user's node did not hold a permission with a lookup key for the portion of name space being referenced. We don't develop this notion further in this paper.

access control functions can now be given (refer to Figure 6-1):

- o lookup.

Given a file name the file system finds the corresponding CE (if one exists). The CE contains sufficient information to access the file since it points to the file's i-node.

- o entry.

There are two types of entry: file creation and file linking. Both result in the addition of new file names to the name space.

File creation adds a new file to the file system as described in Section 5. A correspondence is established between an i-node and the new file, between the (creator's) name for the file and a CE for the file, and between the CE and the file's i-node.

File linking adds a new name for an existing file by establishing a correspondence between the new name and a (new) CE for the file, and establishing a correspondence between the (new) CE and the existing i-node for the file.

The notion of linking introduces a number of new issues. These include charging for files shared by links, deleting files shared by links, and controlling access to files through links. To address these, we introduce the notion of an "owning catalogue entry" (OCE) for a file. Regardless of the number of CE's corresponding to a file, a file has one and only one CE which is its owning catalogue entry. When a file is created, its OCE is set to be the CE corresponding to the name specified by its creator. Any CE for a file which is not the file's OCE is a "link catalogue entry" for the file, or "link" for short.

The system provides an operation for changing the OCE for a object.

There are a variety of approaches to charging for files shared by links. We can suggest two here. One would be to charge the "owner" of the file, where the owner is defined to be the creator of the file's OCE. The other approach would be to distribute the file storage charges, perhaps in a weighted fashion, among the file's owner and creators of link CEs for the file. When a link CE to the file is created or removed, a record could be made in an accounting data base; periodically,

billing software could process this accounting data to determine charges for the shared file. These approaches, as well as others, could be easily implemented; the choice of an approach seems to be more of a policy issue than a technical one.

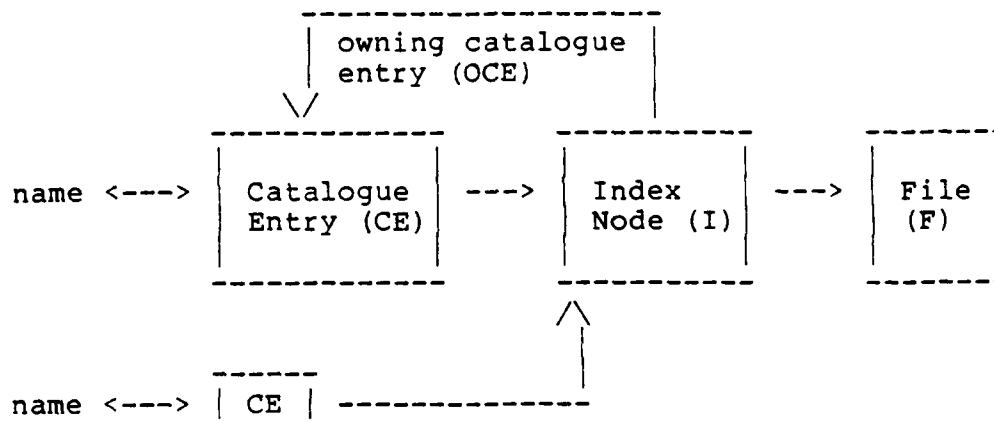


Figure 6-1:

Note, this is a schematic figure. An implementation could incorporate the i-node with the OCE.

o removal/deletion.

The effect of the removal or delete operation depends upon whether the name in question corresponds to an OCE or a link CE for the file.

For a link, the removal operation simply breaks the correspondence between the file name and the link CE, and system removes the CE from the Resource Catalogue.

For an OCE, the removal operation breaks the correspondence between the file name and OCE, breaks the correspondence between the OCE and i-node, and deletes

9

the contents of the file .

After an OCE is removed, links for the file (i.e., link CEs) may remain although the file itself has been deleted. Attempts to reference the file through such links (i.e., by means of the lookup and various access functions) will fail, and an appropriate error

10

indication will be returned . To support this behavior the i-node for a file will remain until the last CE for

11

the file is removed .

- o access control checks.

The discussion of file keys described the access control check function prior to the introduction of links. The question links pose is: should a user's ability to access a file depend upon the file name (CE) used to identify the file? That is, should the key required to access a file depend upon the path through the name space used to specify the file, or should it be more tightly bound to the file itself? The problem with the former approach is that once a link to a file is established, it would be difficult to revoke a user's permission to access the file by means of the link without keeping a record with the file of every link to it. This would be expensive in terms of storage. For this reason access (of a given type) to a file is permitted only if the user holds a permission that is a

9

The other design choice here would be to retain the file contents until the last CE for it is removed. We choose to delete the file because we believe the notion of "file owner" is a useful one, both for accounting and access control purposes. However, we also believe that a viable (but somewhat different system design could be developed from the other design choice.

10

Another reasonable design choice would be to cause read access attempts for the file to fail but to allow write access attempts to succeed and result in the creation of a new incarnation of the file and the establishment of a new file OCE. While this is a reasonable alternative, we don't pursue this idea.

11

This implies keeping a reference count in the i-node that is incremented and decremented on entry and removal operations, and possibly marking the i-node when the file (OCE) is deleted.

key (of the appropriate type) for the file's OCE.

An access control check involves the following steps:

1. the file name supplied by the user is used to obtain a CE.
2. the CE is used to obtain the file's i-node.
3. the i-node is used to obtain the OCE for the file.
4. the user's node is checked for a key of the requested access type for the OCE.

An access control issue remains regarding a user's ability to create links: should an access control check be made when a user attempts to make a link? We think such a check is unnecessary for the uses we envision for NSW since user access rights are checked each time an

12

attempt is made to access the file .

Finally, if it is felt that a user's ability to access a file should depend upon the name (CE) used to identify the file, the access control check procedure specified above could be augmented by requiring the user to hold a lookup key (see earlier footnote) for the CE used to access the file in addition to one for the OCE. We see no need to require this, but the design could support it.

12

However, we see no harm in doing so, and in fact there may be some value in it if NSW were to be used in an environment where it was important to prevent a user from discovering whether a file with a certain name existed. There are a number of alternatives that might be used to accomplish such an access control check; the system might require the user to hold a "lookup" key for the name (CE) used to specify the link target, or it could require the user to have a "lookup" key for the OCE of the link target, etc.

6.2 Set Files

The notion of a "set file" is motivated by a number of factors. These include a desire to provide file system support for programs that automate certain software development, distribution and configuration management tasks, for programs that make use of

partitioned data sets found on some tool bearing hosts¹³, and for flexible, user controllable tool naming and management.

A set file is a compound file. It consists of a collection of members each of which is itself a file. The members of a set file can be manipulated collectively or individually. Set file members can be named individually, relative to their "containing" set file. Consider a set file named a with member files b and c. We write:

a!b

to specify the member b, and

a

14

to specify the entire file set . Should the member b itself be a set file with members d and e, we would write:

a!b!d

to specify b's member d.

To support operations on individual members of set files the lookup function must handle names of member files, and the various access functions must allow access to the set file members.

There are two kinds of set files:

- o NSW set files.

These are set files for which the lookup and various access functions can be performed entirely by and within

13

See Neil Ludlam's paper "UCLA Recommendations on Libraries in NSW", UCLA TR-16, January 18, 1979 for a good discussion of the requirements posed by TBH libraries and partitioned data sets.

14

Since no "\$" is used here, "a" is a partial file spec.

the NSW system. For an NSW set file, data bases internal to NSW and the file itself contain the information required to perform the lookup and access functions. This information is understood by, maintained by, and used by the NSW system.

o Native host set files.

These are set files whose internal structure is not necessarily known to the NSW system. The internal structure is known by the host on which the file resides. A native host set file would be catalogued in the NSW Resource Catalogue but its members would not. Consequently, the lookup and access functions for member files cannot be handled entirely within the NSW system and, in general, would require assistance by constituent host file systems. Examples of native host set files are libraries and PDSs on IBM hosts. NSW workspaces are another example. They are incorporated into the file system by treating them as native host set files (see Section 9).

The resource catalogue model developed in Section 5 must be augmented somewhat to accommodate set files. A set file consists of its members and a "set definition part". The set definition part defines the file's members by defining a correspondence between the member names and the member files.

For native host set files the internal structure of the set file and its set definition part is not known to NSW. For these set files the i-node identifies the set file itself (see Figure 6-2); local host software must be used to manipulate the set definition part and the member files.

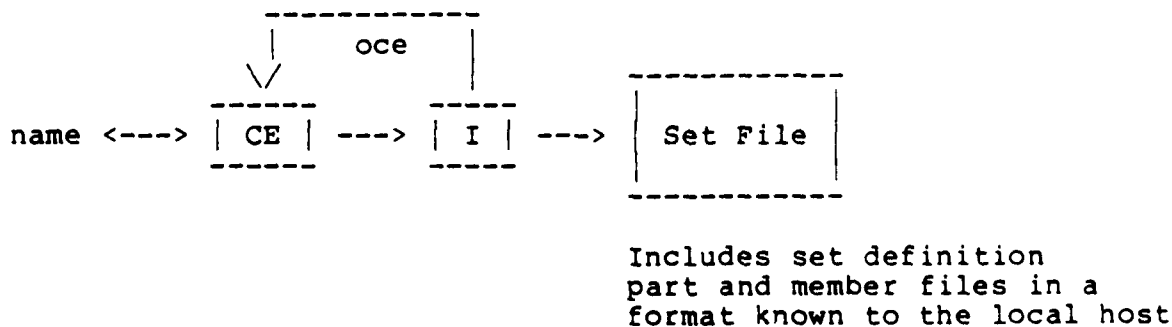


Figure 6-2: A native host set file.

The set definition part of an NSW set file is called the set catalogue, and its internal structure is known to the NSW file system. The i-node for an NSW set file corresponds to the set catalogue for the set file. Conceptually the set catalogue is similar to a collection of catalogue entries in that it defines the names of the member files and identifies the i-nodes for the various member files (See Figure 6-3).

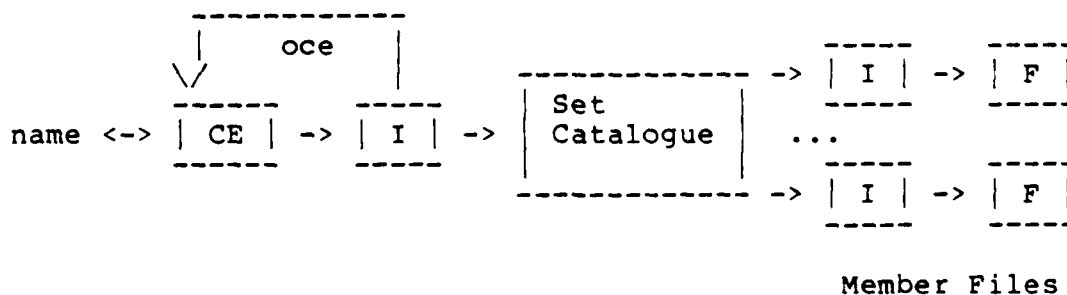


Figure 6-3: An NSW set file.

There is an issue concerning what the OCE of a file which is a member of a set file should be. There are two situations to consider. First, the OCE may be "external" to the set file. That is, the set file contains a "link" to the member; that is, the member is addressable outside of the set, see Figure 6-4. This situation poses no problem since the OCE is just some CE external to the set. A set file which has one or more member files whose OCE is external to the set is called an "open" set file.

The second situation occurs when there is no OCE for the member file external to the file set. There seem to be 4 choices for the OCE in this situation:

1. the member has no OCE.
2. the OCE is the CE of the containing set file.
3. the OCE is the i-node of the containing set file.
4. the OCE is the set catalogue of the containing set file.

We reject choices (1) and (3) for reasons of uniformity and consistency; choice (1) would lead to a system where some files

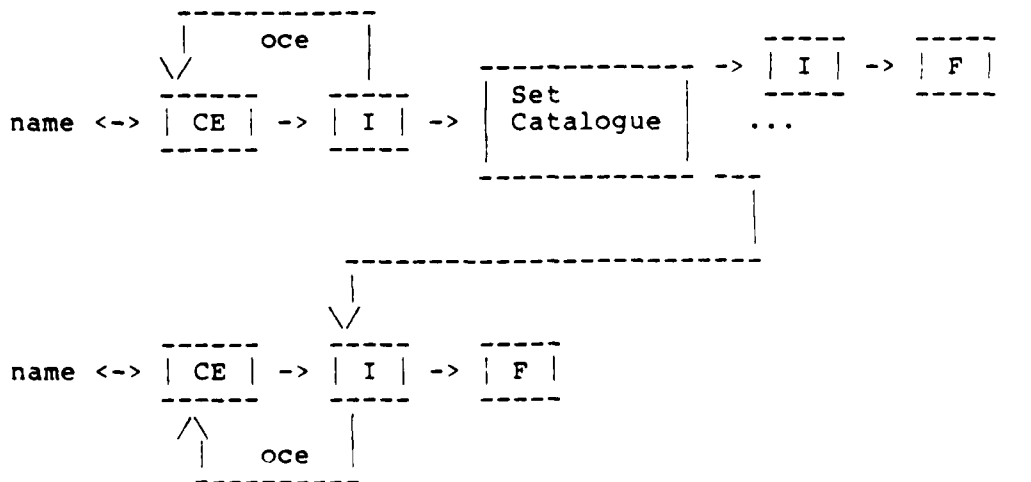


Figure 6-4: An open NSW set file.

had no OCEs, and (3) would lead to one where the OCEs for some files were i-nodes, and for other files were CEs. We choose (4) over (2) mostly to minimize the amount of processing required to manipulate set files and members. For example, choice (2) would require OCEs to be changed when a set file was made a member of another set file.

We call a set file that has no member files with external OCE's a "closed" set file. Figure 6-5 illustrates a simple closed set file. All native host set files can be considered closed.

We can now describe the lookup, entry, removal and access control check primitive functions for set files:

- o lookup.

When a file name specifies a member of a set file, the file system first finds the CE corresponding to the file name. Next it finds the i-node. The i-node identifies the file as a set file. If the file is an NSW set file the file system uses the set catalogue identified by the i-node to find the member file. If the file is a native host set file, the file system calls upon software on the constituent host to complete the lookup function.

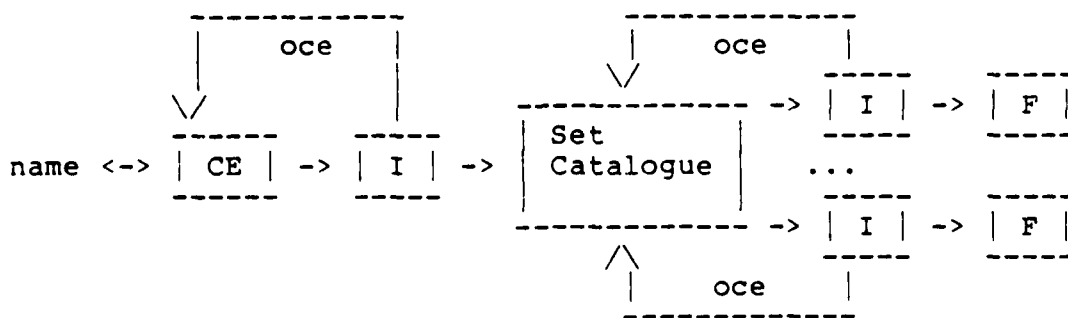


Figure 6-5: A closed NSW set file.

o entry

File system operations exist to create set files and to add members to set files. The creation operation establishes a correspondence between an i-node and an empty set catalogue (and between a name and a CE, and between the CE and the i-node). The "add-member" operation adds a member to the set file by making a new entry in the set catalogue that defines a correspondence between the member name and the i-node for the member file. There are two types of add-member operations. One is similar to establishing a link and results in an open set file. For it, the OCE of the member file is unchanged and remains external to the set file. The other is similar to making a copy, and can be used to create closed set files. For it, the OCE of the member is the set catalogue of the set file. This may be accomplished either by actually copying the file, allocating a new i-node for the copy, and setting the OCE of the copy to the set catalogue, or by simply changing the OCE of the new member file to be the set catalogue. The system should support both operations.

The above applies to NSW set files. The entry function for native host set files requires assistance by software on the hosts that maintain the files.

o removal/deletion.

Removal of a member file breaks the correspondence between the name of the member and the member file by removing the entry in the set catalogue which defines the member name. If the set catalogue is the OCE for

the member file, the member file is deleted.

In addition to removal, it is possible to "extract" a member file from a set file by means of a copy-like operation.

Removal of a member file from a native host set file requires assistance by software on the host that maintains the set file.

- o access control check.

The access control check function for a member of a set file is basically that described in Section 6.1. The ability of a user to access a member file depends on the OCE for the member file and the keys held by the user's node. If the OCE is external to the set file, the access control check is exactly that of Section 6.1. If the OCE is internal, then the OCE of the containing set file is used for the access control check.

Since native host set files are regarded as closed set files, the OCE of the set file is used as the basis of access control checks for members.

There is a design decision to be made concerning the establishment of links to members of closed file sets. At present we see no strong reason, apart from simplicity, for prohibiting them. Of course, links to members of native host set files can not be permitted.

There is another design issue concerned with keys, and whether a key can explicitly specify members of a file set. This would allow a user to selectively control access to set file members. For example, should the system support keys of the form

a!b

which would permit access to member files a!b and a!b!e (assume member b is itself a set file) but not members a!c and a!d? At present, we think the system should recognize keys of this sort.

6.3 File Specs, Scopes, Ellipsis, and the Lookup and Entry Functions

The string specified by a user to reference a file is called a file specification or "file spec" for short. The file system lookup function allows users to reference files with file specs that partially specify file names. Such file specs are interpreted within a context defined by the user's record in the Session Data Base.

This context is defined by a collection of "scopes" that are used to restrict the region of NSW name space considered by the lookup function as it tries to find files (CEs) corresponding to file specs. A scope is a sequence of name components that designates the region of NSW name space that includes names beginning with the scope's sequence of name components. We write:

\$a.b.c*

for a scope with name components a, b, and c. A file whose complete name is \$a.b.c.d.e would be within the region defined by this scope, and a user could use the file spec d.e to reference

15

the file when the scope is in effect .

The system always interprets file specs within the context of the user's scopes unless instructed not to by means of an explicit "unscope" request. As noted in Section 5 the user makes such a request by beginning the spec with the character "\$" which means don't scope; that is, that the string following it is to be interpreted as a complete file name.

When a user logs in, the file scopes in effect for the session are initialized from the user's node record by copying a set of "default" scopes stored in the node record into the user's session record in the Session Data Base. A user can alter the scopes in effect by means of a command which modifies the user's session record. In addition, the default scopes stored in the user's node record can also be changed.

The scope notion should also be used to improve the efficiency of the file name lookup operation. In principle, it should be possible to structure the NSW Resource Catalogue so that lookups within the space designated by the scopes that are in effect require less system resource than those which are not. The NSW 4.1 implementation apparently does not take advantage of this potential for optimization.

NSW 4.1 supports different types of file scopes for different file operations. There are copy, delete and enter scopes. The

15

Whether or not to support file scopes of the form "\$a.b!*" is an issue. Such a scope would allow a user to refer to members of set file a.b without explicitly naming a.b. At present, although we don't feel strongly on this issue, our inclination is to not support such scopes in the interest of conceptual and implementational simplicity.

particular scope used by the lookup operation for a file spec depends upon the file operation being attempted. For example, if the file is to be deleted, the delete scope is used.

The system also allows a user to have more than one scope of each type in effect at a time (an exception here is enter scopes, for which only one may be in effect). In NSW 4.1 when a user has multiple scopes of a given type in effect, they are interpreted in parallel. That is, all regions of name space specified by the scopes are searched, and if more than one match to the scope is found, all are returned to the user. The user is then given the opportunity to disambiguate the spec by indicating the file he meant to reference.

We believe that two changes should be made to the scoping mechanism supported by NSW 4.1:

1. There should be only a single type of scope for file operations. Support for multiple scope types (i.e., copy, delete and enter) serve to complicate and confuse the user model, requiring a user to constantly keep in mind the different scopes in effect and their impact on various commands. For example, if copy and delete scopes are set to different regions, the file spec "a" in the commands:

```
copy (file) a (to file) b
rename (file) a (to file) b
```

refers to two different files.

2. A user should be able to control (by means of appropriate commands) whether multiple scopes are used in a parallel or serial fashion. Currently only parallel use of multiple scopes is supported (See above description). By a "serial fashion" we mean that the multiple regions of NSW resource space should be searched in sequence during a name lookup, and the lookup should succeed and the search cease when a match for the spec is found in a region.

In addition to scopes, NSW provides another feature designed to permit users to reference files using specs that name only some of a file's complete set of name components. Ellipsis ("...") may be used in file specs to signal the omission of zero or more name components. For example, the spec

```
$a.b...e
```

specifies a file whose first two name components are "a.b" and whose last name component is "e", and which has zero or more name components between the "b" and the "e". It is a legal specifier

for the files \$a.b.e, \$a.b.c.e and \$a.b.d.e as well as other files. A file spec may have one or more ellipses in it. The following are all legal file specs:

\$a...b...c
...a.b
\$...a...

When a user supplies a file spec which refers to more than a single file (see ellipsis example above) the system may, depending upon the situation, ask for help to disambiguate the spec so that it refers to a single file. This is typically accomplished by means of the "NSW help" mechanism (See Section 15.3).

We are now in a position to describe the operation of the basic NSW lookup function.

1. If there is a single scope region in effect, use the following Search String Construction Rule to form a search string from the scope and the spec supplied by the user.
 - a. If the spec begins with "\$", then the search string is the spec itself.
 - b. Otherwise, replace the "*" in the scope with a "." and append the spec to the scope to form the search string.

The following examples illustrate this rule:

- o If the scope is \$a.b* and the spec is \$c.d, the search string is \$c.d.
- o If the scope is \$a.b* and the spec is c.d, the search string is \$a.b.c.d.
- o If the scope is \$a.b* and the spec is c...d..., the search string is \$a.b.c...d... .

If the search string contains no ellipses, it is used to search the catalogue for an object whose name exactly matches the string. If an object is found, the lookup succeeds; if not, it fails. If the spec contains ellipses, the string is used to search the catalogue for the set of objects whose names match the search string. If no object is found, the lookup fails. If a single object is found, the lookup succeeds. If more than a single object is found, proceed by means of the help mechanism to disambiguate the spec to a single object.

2. If multiple scope regions are being used in parallel, each scope region is searched as described above, and the set of objects found is formed. If no object was found, the lookup fails. If a single object was found, the lookup succeeds. If more than one object was found, the help mechanism is used to disambiguate the spec to a single object.
3. If multiple scope regions are being used in series, the first scope in the sequence is searched as described above,
 - o if a single matching object is found, the lookup succeeds and searching ceases.
 - o if multiple matches occur within the scope, disambiguation as described above is performed; if disambiguation succeeds, the lookup succeeds and searching ceases, and if disambiguation fails, the lookup operation fails.
 - o if no object match occurs within the single scope, the next scope region is obtained and the above steps are repeated.
 - o if there are no more scopes in the sequence, the lookup fails.

This lookup procedure differs from that currently in use in NSW 4.1 in several ways:

1. All ellipses are explicit. In NSW 4.1 there are implicit ellipses in all file specs. For example the search string for the file spec

\$a.b.c

is

\$...a.b.c...

2. There is only one type of file scope (rather than the three types supported by NSW 4.1).
3. Multiple scopes may be used in a serial fashion in addition to in a parallel fashion.

When entering an object into NSW resource space the user supplies a spec (called an entry name in NSW 4.1) for the object. Part of the entry function, prior to actually entering the object into the catalogue, is to generate a complete name for the object, based on the spec supplied and the user's scopes. The

entry function determines the complete name for the object as follows:

1. Find a set of candidate complete object names by applying the basic NSW lookup function with the following differences:
 - o If no match is found, the set is null;
 - o If multiple matches are found, omit disambiguation and add all matches to the set;
 - o Cease serial lookup when one or more matches are found in a scope region.
2. If the resulting set contains a single object, the complete name of the existing object is used for the entry operation. In this case the user's intent for the entry operation presumably is to replace an existing object. Additional user confirmation could be required and implemented by the help mechanism. Note that specs may contain ellipses, and as long as the result is a single object a unique complete name can be determined.
3. If the resulting set has multiple objects, the entry operation fails.
4. If the resulting set is null, then
 - a. If the spec contained ellipses the entry fails;
 - b. If there is a single scope region, the complete name is the search string;
 - c. If multiple scope regions are being used in series, the complete name is the search string formed from the first scope region;
 - d. If multiple scopes regions are being used in a parallel fashion, disambiguation is performed to determine which search string should be used as the complete name for the entry function.

The motivation for scopes is to make it more convenient for users to reference files in that they relieve them from specifying complete file names. In this sense, the scope notion as specified for NSW is a more general realization of the working directory notion found in many operating systems (e.g., Multics, Unix, TENEX/TOPS-20). It is more general in that multiple scopes may be effect which can be interpreted either in a serial or parallel fashion.

7. Retained Objects: Devices

The basic unit of input to NSW or output from NSW is an NSW file. To support device i/o, the basic idea is that NSW will support the notion of device objects. Device objects are catalogued in NSW name space in the same way other objects are. There is a name which corresponds to a catalogue entry for the device object. The catalogue entry identifies an i-node that contains an descriptor for the catalogued device.

The information recorded in a device object descriptor will include:

- the host supporting the device
- the device type
- device access procedures for
 - read
 - write

There will be an entry or registration function, analogous to the entry function for files. This function is used to enter a device into NSW name space by creating a descriptor for it and adding a catalogue entry for it to the Resource Catalogue.

At the user interface level a "register" command will be provided to make devices known to the system by giving them names in NSW space. That command would interact with the user to gather the information to construct an object descriptor. The interactions would be driven by a "template" for the host and device type. When the necessary information is obtained, the register command will call a Make-Device function which will create a device descriptor, and then call upon the Enter function to create a catalogue entry for the device object, thereby entering the device into NSW name space.

There are several access control issues concerning device objects. The first concerns a user's right to register devices. As described above, registration involves two steps: creation of an NSW device object and entry of a name for the device object into NSW resource space. The second step is adequately controlled by enter keys. Although we see no compelling reason to do so, the first step could be controlled by a permission for device registration.

A second access control issue concerns actually using a device. We believe that some form of access control in the use of certain devices is required. For example, a user shouldn't be permitted to print large volumes of data at an installation where he has made no arrangements for disposal of the printer output. Further work is required to identify the appropriate types of controls required and to design permissions (see Section 10) that implement these controls.

We believe that the most common form of i/o will be printer output. Hence, we will focus on integrating printer devices into NSW resource space in a way that allows users to produce listings of NSW text files.

Input/output with other devices such as tapes, punches, card readers can be handled by the file import/export mechanisms. For example, to write an NSW file to tape, the file can be exported to the host that has the tape drive, and then written to the tape. After exporting the file, the user would make use of local host conventions to actually write the exported file to tape. More integrated NSW support for these other i/o devices can be provided at a later time.

A user would generate a listing by means of a "print" command. For example, the command:

```
NSW: print a.b
```

would cause the file a.b to be printed. The means by which the system determines which printer to use and how to actually have the file printed is the subject of the rest of this section.

Part of a user's session context specifies the line printer to be used when an action requiring a line printer is undertaken and the particular printer is unspecified. The identity of a default line printer for the user is stored in the user's node record in the Node Permission Data Base and is used to initialize the user's record in the Session Data Base at login time.

Commands are provided for the user to change the default printer either for the duration of the current session, by modifying the default stored in the session record, or more permanently, by modifying the default stored in his node record in the Node Permission Data Base. These commands require the user to specify a particular printer. To do so, the user supplies a device spec which is used by the system to search the NSW catalogue for the printer object. The basic NSW lookup function described in Section 6.3 is used for this search with the following modification.

There is a region of resource space named \$Devices* which is a repository of commonly used devices. The basic lookup function using only the user's scopes is performed. If it fails, lookup continues by searching the \$Devices* region.

This, in effect, appends the \$Devices* region in a serial scope fashion to the user's scopes. The net effect of this modification is to cause the system to search a designated system or "public" region when the specified device cannot be found in the user's regions.

A user can cause a listing to be printed by a printer other than his default printer by explicitly specifying the printer object to use. For example,

```
NSW: print a.b  
      (on printer) $Devices.lpt.isie
```

would cause file a.b to be printed on the line printer at the ISIE host (assuming, of course, that the specified device spec actually identifies a registered printer device that is the ISIE line printer).

For line printer devices the only device access procedure of interest is the write access procedure. For printers we identify the following types of write access procedures:

1. Place the file into an area used for spooling listings to the printer on the host that controls the printer. The file might be required to have a special name, and perhaps some special auxiliary information will need to be supplied to ensure that the spooler properly prints it.
2. Move the file by means of FTP to a standard "printer" file at the host that controls the printer. For example, on TENEX/TOPS-20 hosts a file FTP'ed to the file LPT: will be printed on the standard local printer.
3. Use the "TIP copy" mechanism to cause the file to be printed on a particular ARPANET TIP. This generally involves placing the file, along with information that specifies a TIP and a TIP port, in a host area used by a TIP printer spooler.
4. Activate a procedure on a specified host, supplying it the file or sufficient information about the file to enable it to access the file. The procedure will take responsibility for printing the file.

In each of the above cases, the files might have to be translated into a "print" format suitable for outputting to the particular line printer device.

In addition, since NSW users will not have direct physical access to all printer devices, it may be necessary in some cases to include external information with the file which can be interpreted by the operator of the printer as routing instructions (e.g, where to send the listing after it is removed from the printer).

The first three access procedures are common ones. The fourth

provides an extension mechanism for introducing printers which use different protocols.

8. Retained Objects: Services

One of the major features NSW provides users is the ability to access computational services on hosts. Services are a type of object managed by the system. This section considers service objects, and the next section considers the related notion of workspace objects. The two sections should be read together since service and workspace objects are closely related.

The basic idea is that a user may invoke a service by name, or more precisely by "service spec". The subject of this section and much of the next one is how the system supports services for users.

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

1. Single interactive programs. The user interacts with a single program on a "service bearing host". This is the principal type of service NSW 4.1 supports.
2. Service bearing host NSW command interpreters. The user interacts with an NSW style command language interpreter on the host, presumably to manipulate files and to run programs on the host. We shall call such a command interpreter a "workspace command interpreter" (WS-CI) because it operates in the context of a workspace on the host (see below).
3. Service bearing host command interpreters. The user interacts with the standard command language interpreter for the host, again, presumably to manipulate files and run programs on the host.
4. Service bearing host operating systems. The user interacts directly with the service bearing host operating system.
5. Batch Services. The user makes use of NSW features to submit a job to a batch processing host.

The first 3 types of service bear a number of similarities and it is difficult to make a clear distinction among them. For each, the result of invoking the service is an instantiation of the service in a "workspace". From the point of view of the NSW system, a service bearing host command interpreter (case 3) is just a particular single (albeit somewhat complex) program. The following discussion merges cases 1 and 3, and refers to these services as "program" services. We shall see below that the program service and WS-CI service notions are also related.

The result of invoking a service of the fourth type is an

instance of the service where the user must interact with the operating system in a very "native" mode. There is no NSW workspace, the user is not logged in, etc. In fact, a service of this type would be instantiated by means of a standard ARPANET initial connection protocol (ICP) exchange with the "logger" socket at the host, the standard contact point for terminal access to the host. With this observation, we generalize the fourth service type to include any service that can be reached by an ICP exchange with any contact socket at a host. We call services of this type "service bearing host ICP" services, or ICP services for short.

Services of the fifth type are called "batch" services. The interactive batch submission (IBS) package currently supported by NSW 4.1 is an adequate basis for batch services in the system. The IBS package can be regarded as a program service (type 1 above) which is used to initiate batch services (type 5 above).

In summary, we have identified four service types:

- program services
- WS-CI services
- ICP services
- batch services

A user should be able to invoke a service by the "use" command regardless of its type. For example, suppose the operating system for the RADC-Multics host was an ICP service catalogued in NSW name space as "Rome-Multics". The command:

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

should place the user in contact with the Multics operating system at Rome. Similarly, suppose the compiler for the BCPL programming language is catalogued as the program service "bcpl". The command:

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

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

The service spec supplied by the user as an argument to the "use" command is used by the system to search the NSW catalogue for the service object. The system uses a slightly modified version of the basic NSW lookup function described in Section 6.3 that is similar to that used for device lookup:

There is a region of resource space named \$Services* which is a registry of commonly used services. The basic

lookup using only the user's scopes is performed. If it fails, lookup continues by searching the \$Services* region.

In effect this appends the \$Services* region to the user's scopes in a serial fashion. The net effect of this minor modification to the basic lookup function is to search a designated system or "public" area when the specified service can not be found in the

16

user's area.

Depending upon the type of service in use, a user might find himself interacting with one of three conversational partners:

1. NSW command interpreter (NSW-CI).

The NSW-CI is the user's initial conversational partner

17

when he accesses the system. As described above, other conversational partners are created by commands to the NSW command interpreter.

2. A service.

Service conversational partners are created by the "use" command. The conventions used within a service for naming objects, such as files, vary from service to service and host to host (see below). (In the case of a batch service, the conversational partner is the interactive batch submission (IBS) program.)

3. Workspace Command Interpreter (WS-CI)

A WS-CI may be instantiated in two ways. One way is by direct invocation through the NSW-CI by a "use" command. The other way is indirectly. On some hosts program services are instantiated with a companion

16

A number of refinements could be made to this rule, including allowing a user to control whether the \$Services* region is searched prior to or after his own scope regions, allowing multiple public service regions, etc. Relatively minor implementational changes would be required to support these refinements. However, we don't consider them further in this report.

17

Other ways of accessing the system are possible, and will be discussed later.

WS-CI. In these cases, when the conversational partner is the program service, the companion WS-CI can be invoked by means of a special control character. WS-CIs implement a workspace command syntax which is standard across hosts, and they use naming conventions which enable a user to reference both NSW and native host objects (see below).

The user may, of course, control which of these is his conversational partner at any time by means of NSW commands and control characters.

Figure 8-1 illustrates the relationship of these conversational partners for a program service and a companion WS-CI, and the possible transitions among them.

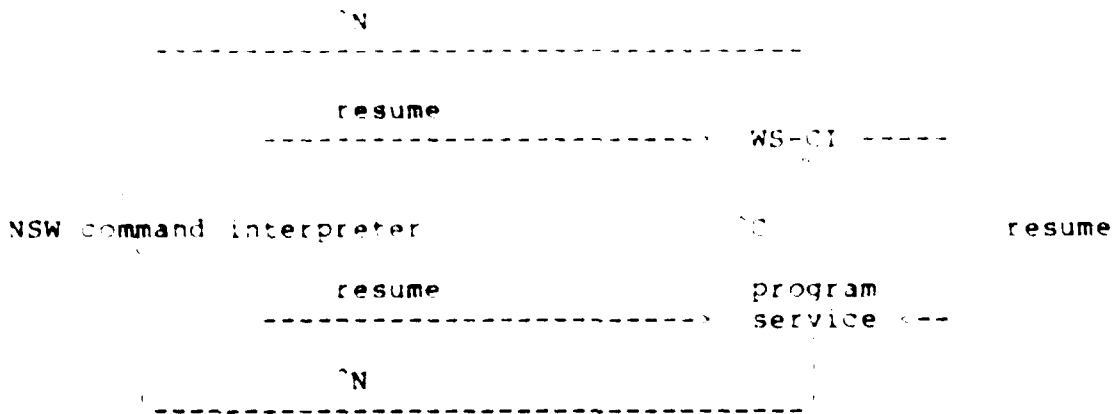


Figure 8-1: Conversational partners for a program service.

There is a standard NSW control character, shown as ^N in the figure, used to change the user's conversational partner to be the NSW command interpreter. The "resume" command can be issued to the NSW-CI to change the conversational partner back to the service instance. NSW permits a user to have more than one service active at a time. To distinguish among them, the resume command accepts a service instance name (see Section 4) as a parameter.

Not every host will support the notion of WS-CIs as companions to program services. For those that do, there is a standard control character, shown as ^C in the figure, used to switch conversational partners from the program service to the WS-CI.

For those that do not, when the ^C character is typed it will be passed directly to the service program. When the conversational partner is a WS-CI, its "resume" command can be used to switch back to the service.

The situation is somewhat simpler for ICP services since they have no WS-CI. Figure 8-2 illustrates the conversational partners for an ICP service. It also serves to illustrate the situation that occurs when the service is a WS-CI (i.e., a WS-CI without a companion program service), or when it is a program service without a companion WS-CI.

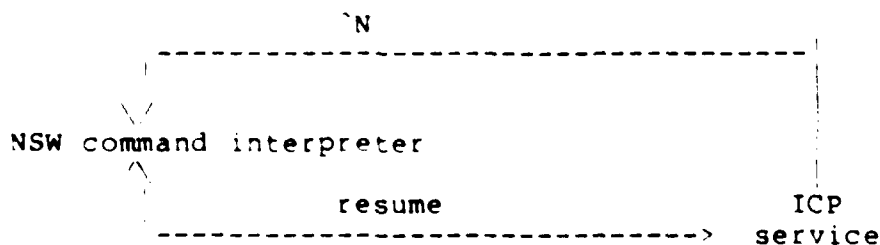


Figure 8-2: Conversational partners for an ICP service.

The function of a workspace command interpreter is to provide means by which a user can manipulate objects, such as files and programs, in a workspace. Although implemented for a number of different hosts, there are NSW standards for WS-CIs:

1. WS-CIs implement a standard, simple command syntax which is uniform for all hosts.
2. WS-CIs implement a standard set of functions which is the same for all hosts. These functions include
 - o Interrupt Program (via a standard interrupt character - ^C)
 - o {Start / Kill / Resume} Program
 - o {Get / Deliver} File
 - o Quit (i.e., stop WS-CI)
 - o NSW / Native (file mode) on / off
 - o Logout of NSW

3. WS-CIs use a distinguishable prompt. so that the user can tell which conversational partner (WS-CI, NSW-CI, service) he is interacting with. We suggest that the service instance name be used as the WS-CI prompt.

Another aspect of NSW services that reference files is the mode of file access provided. A number of file access modes are possible. These include two "pure" modes:

1. NSW access mode.

NSW file conventions are used while interacting with the service. File specs supplied by users are interpreted using the NSW Resource Catalogue.

2. Native access mode.

The file conventions of the service bearing host operating system are used. File specs supplied by users are interpreted within the context of the local host file system.

Some services may provide a mixed mode where a user may indicate to the service by some syntactic convention the type of file (NSW, native) intended when a file spec is supplied. Mixed mode access is described in more detail below for program and WS-CI services.

In general, NSW file access mode requires NSW supporting software on the service host which the service interacts with during file references, whereas native mode does not. The interactions between the service and the supporting software may be explicit or implicit.

ICP services operate in a purely native mode. Consequently, the only file access mode provided for them is native. For batch services, after job submission the batch program itself runs under the control of the batch processing system on the batch service host in a native mode. Of course, it is desirable to be able to include NSW files in batch runs. The batch submission system (IBS) interacts with a user to determine the files to be used in a batch run so that they can be submitted with the batch job to the batch processing host. Consequently, IBS must support NSW file access mode. It is conceivable that IBS could also support native or mixed file access modes.

A capability provided by WS-CIs is the ability to move files between NSW file space and workspaces. When interacting with a WS-CI means are required to differentiate between files in the workspace and files in NSW resource space. That is, a mixed file mode is required where both NSW files and workspace files can be referenced. For this purpose, a special character will be used

as a syntactic device for signalling an NSW file name. The special character either will appear as the first character in the name (to indicate a name in NSW space), or it will be absent (to indicate a name in the workspace). For example, suppose the character were "^". The following would be legal names:

```
a.b      ;absence of ^ indicates that the name should be
          ;interpreted as that of a workspace file.
^a.b     ;^ signals that the NSW Resource Catalogue and
          ;scopes should be used to interpret the name.
^$a.b    ;^ indicates the name is an NSW name, and $
          ;signals NSW that none of the scopes in
          ;effect should be used.
```

These naming conventions represent a simple mixed mode between the extremes of pure NSW and pure Native modes for which the file name syntax implicitly specifies the mode desired.

Another issue is whether the syntax for names of workspace objects, such as files, should be uniform across all host types, or whether each WS-CI should use its own syntax for workspace files. We believe this should be an option of the host.

For program services the file access modes supported depend upon the program service itself and the NSW supporting software provided by the service host. Some services may support only NSW file access mode. This is the file access mode supported for the encapsulated tools of NSW 4.1. All user supplied file specs are interpreted within the context of the Resource Catalogue and the file scopes in effect. Other services may support only native access mode. In order for these services to be able to access NSW files, copies of the files must be moved into the service workspace where they can be accessed in a native mode. The interactive tools supported by NSW 4.1 on the UCLA IBM host operate in this manner. The movement of files from NSW space to the workspace can be done either by a companion WS-CI in the workspace with the service, or by commands to the NSW command interpreter. Finally, some services may operate in a mixed mode similar to the one described above for WS-CIs.

Table 8-1 summarizes the types of file access modes that may be supported with the four service types.

Service Type	File Access Mode		
	NSW	Native	Mixed
program	yes	yes	yes
WS-CI	yes	yes	yes
ICP	no	yes	no
Batch			
IBS	yes	no	no
batch prog	no	yes	no

Table 8-1:

The types of services and modes of file access possible for services have been discussed. We turn now to the means by which service objects are entered into NSW resource space. As with other types of objects this is a two step procedure. First, a descriptor for the service object is created, and then an entry for it is made in the NSW Resource Catalogue. The Resource Catalogue binds the object to its NSW name by means of a catalogue entry, which holds the name, and an i-node, which holds the descriptor.

The information stored in the i-node for a catalogued service is, in effect, a specification of the procedures for instantiating and controlling the service. The type of information stored in the descriptor depends upon the service type.

For an ICP service the information required is simple: the host that provides the service, and the ICP contact socket at that host. For example, the following dialogue would enter an ICP service into NSW name space.

```
NSW: register service
      NSW name for service: rome-multics
      Service type: icp
      Host: radc-multics
      Contact socket: 1
      ICP to RADC-MULTICS Socket 1 registered as
      $Services.rome-multics
```

For this example we have assumed that the scope in effect is "\$Services*".

As noted above, not all hosts will provide WS-CI service. For each host that does, there will be a "pre-defined" service

descriptor for its WS-CI and the WS-CI will be "pre-entered" into NSW name space. Names for the WS-CIs will be chosen by system administrators, and are likely to be in the public "\$Services*" region of name space.

The names chosen are not important for two reasons. First the link mechanism allows users to choose their own names for the WS-CIs. Second, the NSW command interpreter provides a "ws-ci" (use workspace) command which allows a user to use the WS-CI in a workspace without naming the WS-CI.

Somewhat more information is required to create a service descriptor for program services.

NSW 4.1 stores the following information about interactive tools (program services) in tool descriptors.

1. The host(s) that provides the service.
2. The host's name for the service. (i.e., file name for the executable)
3. The tool's NSW properties. This includes information such as: the type of communication path to the user (TELNET or binary) the tool expects; the tool termination procedure, including the disposition of any workspace files it creates.
4. Host specific data. This data is uninterpreted by NSW core software. It is passed to the NSW support software on the host that supports the service when an instance of the service is started. Currently this information is used in two quite different ways by tool bearing hosts (TBHs).
 - a. For some the data is used to specify files used by the tool that are local to the host. This enables NSW support software on the TBH to determine which file references made by the tool are for NSW files and which are for local files.
 - b. For other hosts the data specifies commands for what is, in effect, a "tool bearing host command interpreter" implemented by NSW supporting software on the host. NSW support software activates the TBH command interpreter prior to tool startup and at tool termination. The commands specified in the tool descriptor define a set of interactions between the TBH command interpreter and the user in terms of queries for the user and expected user responses. Such interactions are typically used to obtain the

names of files used by and produced by the tool.

5. Tool entry point information. For example, entry points for warm start, tool cleanup and termination are currently defined. (NSW 4.1 does not currently make use of this information.)

Notice the similarity between the TBH command interpreter and the WS-CI notion. The TBH command interpreter is, in effect, a WS-CI that is automatically activated by NSW support software at tool startup and termination.

Program services are somewhat more general than NSW 4.1 tools in two ways. First, a program service may be implemented by an NSW file or a native host program. NSW 4.1 tools are limited to native host programs. Allowing NSW files to be used as services makes NSW an easily extensible system. It facilitates the use of NSW for developing, debugging, testing and evaluating, and releasing new program services. The second way is that a program service can operate in any of the file access modes described above. NSW 4.1 tools are limited to NSW file access mode.

From this, we believe that the information that should be stored as part of an object descriptor for a program service object is:

1. The host the object runs on.
2. The name of the executable program that implements the program object.

There are two types of programs: programs that are NSW files and programs that are native host programs. A program that is an NSW file is one that is stored in the NSW file system. A native program is one that is not in the NSW file system. The name of a native program is a name interpretable by the host.

3. The file access mode of the program service (NSW, Native or Mixed). In some cases, the mode could be don't care. That is, some services may be able to operate in any of the modes. Of course when the program is instantiated a mode must be specified.
4. Information used by NSW supporting software on the program host to manage the program service.

The following example illustrates registration of a program service.

AD-A185 967

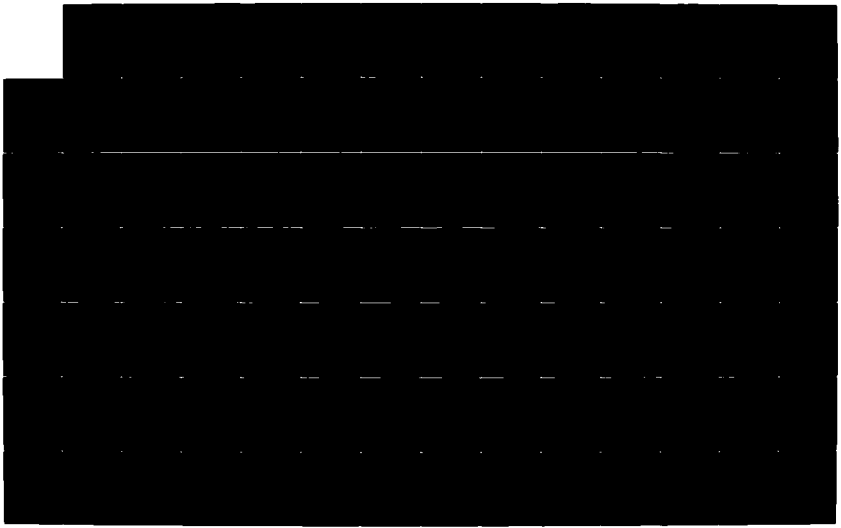
NSW : NATIONAL SOFTWARE WORKS / PERFORMANCE ENHANCEMENTS
(U) BOLT BERANEK AND NEWMAN INC CAMBRIDGE MA
R E SCHANTZ ET AL. JAN 81 BBN-4600

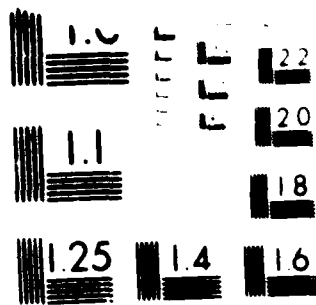
2/4

UNCLASSIFIED

F/G 12/5

NL





U.S. GOVERNMENT PRINTING OFFICE

NSW: register service
NSW name for service: bcpl
Service type: program
Host: radc-20
Native or NSW program: native
Program name: <bcpl>bcpl.exe.2143
Connection type: telnet
File access mode(s): NSW
 . (Specification of host
 . dependent information
 .
Program <bcpl>bcpl.exe.2143 at RADC-20 registered as
\$Services.bcpl

The collection of host dependent information will be driven by a template supplied by host software personnel. The template will be a specification of questions for the user and corresponding expected user responses. It will be used much as the "canned" commands for TBH command interpreters that are stored in some NSW 4.1 tool descriptors.

Users are likely to occasionally make errors when entering
18
program services into NSW resource space . The system provides means to correct errors of this sort by modifying the descriptor for a service. The following scenario illustrates how this might be done.

First a user registers a file as a service under an experimental name in order to check out the registration parameters. After modifying the parameters, the user renames the service, presumably prior to giving other users access to it.

18

The errors we mean here are errors in specification of object descriptor information, not program bugs. We assume the user has other means at his disposal to diagnose and correct program bugs.

```
NSW: register service
      NSW name for service: $BBN-NSW1.Programs.srccom-exp
      Service type: program
      Host: radc-20
      Native or NSW program: NSW
      Program name: $BBN-NSW1.Programs.srccom.obj
      Connection type: telnet
      File access mode(s): NSW, native
        . (Specification of host
        .   dependent information)
        .
      Program $BBN-NSW1.Programs.srccom.obj registered as
      $BBN-NSW1.Programs.srccom-exp
```

The user next tests the registration specification, finds a few errors, and corrects them by means of the modify command.

```
NSW: modify service srccom-exp
      Modify type? no
      .
      .
      .
      Modify ...
      .
      .
      .
      Program $BBN-NSW1.Programs.srccom-exp modified
```

After more testing the user is satisfied with the specification and renames the service.

```
NSW: rename srccom-exp srccom
```

The registration process for entering a batch type service into NSW resource space involves specifying information necessary to permit an interactive batch submission (IBS) program to initiate the batch service. This would, in effect, involve specification of the necessary job control information for the service, as well as specification of the IBS-user dialogue required to gather the job specific information necessary for each job submission. This information would then become part of the object descriptor for the batch service.

One way this could be done would be to enter the necessary job control information and IBS specification into NSW files, and specify them when the service is registered. This is illustrated below

NSW: register service
NSW name for service: FORTRAN
Host: ucla-ccn
Job control file: FORTRAN.jcl
IBS specificatio file: FORTRAN.ibs
Batch service at UCLA-CCN registered as
\$Services.FORTRAN

The "register" command provided by the NSW command interpreter is supported by a set of functions implemented by NSW core software that create object descriptors for the various service types. There are Make-ICP, Make-Program and Make-Batch functions which create descriptors for ICP, program, and batch services, respectively.

Make-ICP (Host, Contact-Socket)
-> service-descriptor

Make-Program (Host, NSW-or-Native-Program, Program-Name,
NSW-Properties, File-Access-Mode,
Host-Specific-Info)
-> service-descriptor

Make-Batch (host, ...) -> service-descriptor

These operations accept information gathered by the NSW command interpreter and create object descriptors of the appropriate types that can be entered into the NSW Resource Catalogue.

Finally, there is the issue of who can register services. As with other NSW objects, service registration can be controlled, in part, by enter keys, which serve to control the regions of NSW name space a user can access to make entries. Whether additional control of service registration is required and the nature of those controls, if any, is largely a policy issue.

The NSW permission system (see Section 10) is flexible enough to support permissions to control service creation that are as coarse or as fine as policy requires. For example permissions could be defined for:

1. The creation of "external" services. These are services implemented by native programs.
2. The creation of "internal" services. These are services implemented by NSW files.
3. The creation of ICP-services.
4. The creation of batch services.

9. Retained Objects: Workspaces

Workspaces have been mentioned in Sections 4 and 8 in connection with program and WS-CI services. Workspaces as NSW objects are discussed in somewhat more detail in this section.

The notion of a "workspace" was initially developed to support program execution. An "area" was required where an execution context could develop for a running program. An assumption was made that not all users would be assigned "user slots" on all hosts for which they might have permission to use programs. The idea was that the workspace area would provide an execution environment through which a program could access NSW resources and modify NSW files.

As part of the "use" operation for a program service (See Section 8) NSW obtains a workspace and places the program into execution "within" it. Copies of NSW files would be moved into workspaces where programs could access them, and new files could be created within workspaces for subsequent delivery into NSW file space.

The workspace notion leads to two levels of file system. A user, or a tool acting on a user's behalf, may manipulate files either within the NSW file system or within a program's workspace. When a program accesses an NSW file, a copy of it is moved from NSW file space to the program workspace, and when a program creates a new NSW file a copy of it is moved from the program workspace to NSW file space.

Although workspaces are implemented in NSW 4.1, they are not part of NSW resource space but rather are treated as ephemeral objects. Treating workspaces as retained objects, such as files, will result in improvements in:

1. Performance.

The workspace notion as embodied in NSW 4.1 limits the lifetime of a workspace to the duration of the program whose execution it was created to support. By treating workspaces as retained objects and providing user level operations for them, features such as permanent workspaces, program chaining, and seeded workspaces can be supported. A permanent workspace is one which would exist for a relatively long time (e.g., across user sessions) as opposed to the duration of a single tool session. Use of a permanent workspace can result in performance improvements since the results of running programs can be allowed to accumulate in the workspace and be available for subsequent programs without requiring that they first be moved back to NSW file space and then retrieved. This can result in

significant performance enhancements for a user whose needs are primarily satisfied by a single host. Program chaining is the use of the same workspace to support a sequence of program services for a user rather than allocating a new workspace for each program as is the current practice. Performance improvements result because repeated workspace allocation and deallocation need not be done, and as with permanent workspaces, the results of previous programs are immediately available to later programs run by the user. A seeded workspace is one which has NSW files moved into it prior to tool execution.

The notion of workspace command interpreters (WS-CIs) introduced in Section 8 provides some of the capabilities needed to support chaining, seeding, and permanent workspaces.

2. User model of the System

The user's conceptual model for NSW is somewhat confused because of the way NSW 4.1 treats workspaces.

In NSW 4.1 the distinction between the two levels of file system is not emphasized. Indeed, because files migrate automatically between NSW file space and tool workspaces (by means of NSW supporting software on the hosts), a user, in principle, does need not be aware of the distinction. In practice, there are situations in which the two levels are visible. For example, the system allows a user to have simultaneous active tool sessions. In these situations, files created by one tool are not accessible by another tool until they are delivered from the workspace into NSW file space. Because newly created files are not delivered immediately, a naive user may be surprised that a file created by one tool may not be immediately accessible to another. Even for users who understand the two levels of file system and how NSW 4.1 treats workspaces, the system provides no way to control the movement of files between workspaces and NSW file space.

By treating workspaces as full NSW objects and providing user operations for them as objects, a more consistent, understandable user model can be supported.

The following paragraphs discuss workspaces as NSW objects. As noted in Section 8, it is difficult to discuss workspaces separately from services. Familiarity with Section 8 is therefore assumed.

Like any other object, workspaces may reside in NSW name space. A workspace may have a name, which is defined by its catalogue entry in the Resource Catalogue, and an i-node, which holds a workspace descriptor for it. We shall temporarily defer a discussion of how workspace objects are entered into NSW name space.

NSW workspaces can provide temporary or long term file storage. In this regard, they exhibit properties similar to those of set files (see Section 6.2). NSW files can be moved into workspaces, and workspace files can be moved into NSW file space. These are similar to the "add member" and "extract member" operations for set files, and are typically used to "get" NSW files for program services that run in the workspace, and to "deliver" the results of program services back to NSW file space. Because of these similarities, we shall use the syntactic conventions introduced in Section 6.2 for referring to workspace files. Since the manner in which files that are in a workspace are managed is outside of the control of NSW core software and are the responsibility of software on the workspace host, workspaces are more like native host set files than NSW set files.

The following two examples, which move files between NSW space and a workspace, illustrate the similarity between workspaces and set files. In them, we assume the user has access to a workspace which the scopes in effect allow him to refer to as "r20".

NSW: copy (file) r20!a.b (to) c.d

This operation moves a copy of the file a.b which resides in the workspace known as r20 into NSW file space as a file named c.d.

NSW: copy (file) g.h (to) r20!i.j

This operation moves a copy of the NSW file g.h into the r20 workspace as a file named i.j.

The examples above were commands to the NSW command interpreter. The same operations could be initiated through a

19

WS-CI running in the workspace . The following WS-CI commands have the same effect as the above NSW-CI commands.

WS: copy (file) a.b (to) ^c.d

WS: copy (file) ^g.h (to) i.j

19

Assuming, of course, that the host supports WS-CI services.

Here the "WS:" is the WS-CIs prompt. Recall from Section 8 that "^" is used in the mixed file access mode by WS-CIs to signal an NSW file name.

A workspace can be named in a "use" command: When named in this way, the system will run the specified program service in the workspace. For example, the following command would result in the creation of an instance of the bcpl service in the r20 workspace.

20

```
NSW: use (service) bcpl,
in (workspace) r20
...
```

To implement this command, the system checks the Resource Catalogue, using the scopes in effect, for a service whose spec

21

is bcpl and a workspace whose spec is r20 . It checks that the user has permission to run the bcpl program service and to use the r20 workspace, and that both reside on the same host. Then it interacts with NSW supporting software on the host to start up the bcpl program in the r20 workspace.

After a program service has been started on a host that supports WS-CIs, the user can activate a companion WS-CI by typing the special WS-CI activation control character (see Section 8).

On hosts that support WS-CIs, a WS-CI can be started in a workspace (without a companion program service). For example, the following starts an instance of a WS-CI named WS-CI.radc-20 in a workspace known to the user as r20.

```
NSW: use (service) WS-CI.radc-20,
in (workspace) r20
...
```

Since running WS-CI services is likely to be a common operation, the NSW command interpreter provides a command that starts a

 20

Other syntactic mechanisms could be used to signal the specification of a workspace. We use the "command modifier" mechanism used in the Unix based NSW command interpreter for NSW 4.1.

21

The lookup rule for services was described in Section 8. The basic NSW lookup rule described in Section 6.3 is used for workspaces.

WS-CI in a specified workspace. The command

```
NSW: ws-ci (in workspace) r20
...
```

is equivalent to the immediately preceding use command.

If no workspace is specified when a user attempts to start a program service, the system selects a workspace for the program. The manner in which this is done depends upon whether the user

22

has a permanent workspace on the program host. If so, NSW uses the permanent workspace for the program. If not, it allocates a workspace from a pool for the program, and gives it a name in a region of NSW name space accessible to the user. This is done to enable the user to move files between NSW space and the workspace (as described earlier) by means of NSW commands that name the workspace. That is, it is entered into NSW name space. An issue concerns whether a user can retain the use of a temporary workspace across services instances or across NSW user sessions. A reasonable approach seems to be to allow a temporary workspace to remain allocated to the user for the duration of his NSW session, but to have it deallocated when he logs out. This can, of course, be relaxed later to allow for longer term allocation of temporary workspaces. Thus, the user can manipulate the workspace until he explicitly terminates it, or logs out.

To determine whether the user has a permanent workspace on the program service host, NSW core software consults the session record in the Session Data Base. A record of a user's permanent workspaces is maintained in the Node Permission Data Base with the user's node record. At login time permanent workspace information is moved from the user's node record to the user's record in the Session Data Base.

Users can obtain permanent workspaces in one of two ways. The first way is to be assigned one from a pool of workspaces managed by NSW. This is a two step operation. First, the user is granted the permission to have a permanent workspace on the host by another user authorized to assign permanent workspaces. Next,

the user exercises that permission to assign himself the

22

And unless otherwise requested.

23
workspace .

The following example illustrates this procedure. First, the
24
user is granted permission to have a workspace on host radc-20.

```
NSW: grant permission
      Node name: Thomas
      Granting to Thomas.Robert.H.1
      Permission: ws
              for host: radc-20
      Permission:
      The following permission:
      [ws, radc-20]
      added to node for
      Robert H Thomas
```

Next, the user exercises this permission and obtains the workspace by means of the "assign" command.

```
NSW: assign workspace
      NSW name for workspace: r20
      Host for workspace: radc-20
      Workspace on RADC-20 assigned as $Thomas1.r20
```

Here the scope in effect is "\$Thomas1", and as a result the workspace entered into NSW name space with the name \$Thomas1.r20.

The second way a user can obtain a permanent workspace is to register an external user slot. The idea here is that the user can make an external account on a host known to NSW for subsequent use as a workspace. This would be useful to users with host accounts apart from NSW who would like to access those user slots, or the files they contain, through NSW. After registering such a user slot, the user can use it through NSW as a workspace or continue to use it directly through normal host access mechanisms externally to NSW.

As with the registration of other objects such as services and devices, NSW accomplishes workspace registration by first

23
The operation is done in two steps to avoid the access control problem that would result by requiring that the granter of the workspace hold an "enter" permission for the grantee's region of name space

24
The permission system is described in Section 10.

creating an object descriptor for the workspace object and then entering it into NSW name space. The following illustrates registration of an external user slot as an NSW workspace.

```
NSW: register workspace
      NSW name for workspace: ie
      Host for workspace: isie
      Directory name: BTHOMAS
      Login directory? yes
      Password: rht
      Require password to access? yes
      Workspace on ISIE registered as $Thomasl.ie
```

As before, the user's scope is assumed to be "Thomasl". Some of the dialogue is driven by a host dependent template similar to the templates used for the registration of program services (see Section 8). The user is asked to supply a password for the user slot to permit NSW to verify his right to access it. The intent is that NSW will not store the password, but rather pass it on to the workspace host to check it. We assume that NSW supporting software on the host has sufficient capability to access the user slot in order to start services in it and manipulate files in it without a password. To provide an extra measure of security the user can specify that a password check is to be performed each time the workspace is accessed. The intent is that NSW supporting software on the host will deny access to the workspace

25

from NSW until a valid user slot password is supplied .

NSW permits the same workspace to be used for multiple simultaneous instances of program and WS-CI services. However, since not all users may want such behavior, the system provides workspace attributes for controlling simultaneous use of a workspace. A workspace attribute is part of the workspace descriptor maintained with the i-node for the workspace. It can be set to prohibit simultaneous use, to permit simultaneous use only by a particular node or set of nodes, or to impose no restrictions on simultaneous use. The use of all workspaces is,

25

The intent is that the NSW supporting software would pass the password supplied by the user at workspace access time to host operating system software to be checked. If our assumption that the NSW supporting software has sufficient capability to access a workspace user slot without a password does not hold for a particular host, a password for the user slot will need to be supplied and checked in this fashion each time the workspace is accessed, regardless of the user's answer to the "require password" question.

of course, controlled by the permission system.

Next we turn to a discussion of how NSW implements the workspace notion. As the discussion above has suggested, implementation of the NSW workspace concept is based on host user slots. A user slot is a host object, variously called a user account, login account, or login directory. Its implementation varies from host to host.

NSW software makes use of host user slots for several purposes:

1. To implement permanent workspaces. Here we require that each permanent workspace be implemented by a host user slot that is dedicated to the permanent workspace. That is, hosts that permit permanent workspaces must support long term user slot to workspace bindings.
2. To implement temporary workspaces. As previously described, temporary workspaces are required to support the instantiation of program services on hosts where users do not have permanent workspaces. It is conceivable that some hosts may be able to use a single user slot to support more than a single temporary workspace.
3. For the storage required by files.
4. For general NSW use by NSW supporting software on the host. Here the host would be free to utilize user slots as it sees fit either to implement workspaces or as storage for NSW files.

Of course, not all hosts may choose to, or be able to, support all of these uses of user slots.

As noted above, a workspace is an NSW object similar in many ways to a user slot. An important difference is that workspaces exhibit a set of properties that are uniform from host to host, whereas the properties of user slots vary from host to host. This uniformity of workspace properties is achieved by NSW software. The properties that are uniform from NSW host to NSW host are, in effect, defined by the NSW functions that can be performed on workspace objects. Most of these properties have been discussed earlier in this section.

Figure 9-1 illustrates the relation between NSW workspaces, host user slots, and the host software that implements workspace properties.

Because treatment of workspaces in the fashion described in this document is a significant departure from the treatment of workspaces in NSW 4.1, an appendix discusses in considerable

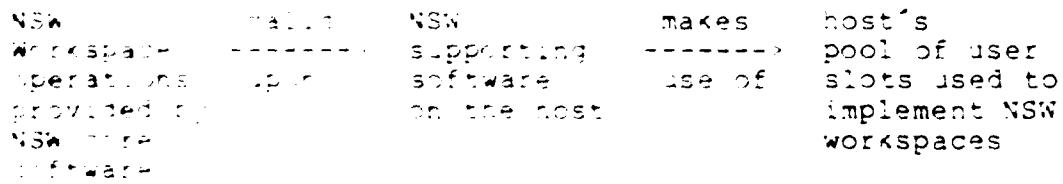


Figure 9-1:

detail how user slots are used to implement workspaces. A reader not interested in these details may skip that appendix.

10. Nodes and Permissions

This section first presents the notion of permissions as an approach to access control, and then develops a permission system for NSW. The permission system presented is based on ideas described by Warshall in a note titled "On Names and Permissions" dated February 2, 1979.

10.1 Rules for Permissions

As mentioned in Sections 3 and 5, nodes are NSW's internal representation for users, and permissions define a node's (user's) rights to access objects.

Formally, a permission (p) is a tuple:

$$p = \text{node}(p), \text{right}(p), \text{ancestor}(p).$$

26

which authorizes $\text{node}(p)$ to exercise $\text{right}(p)$. For example, $\text{right}(p)$ could be the right to read files in a certain region of NSW name space. $\text{Ancestor}(p)$ is the ancestral permission which was exercised to create permission p (see Rule 1 below).

A relation of dominance is defined over pairs of rights. The relation $r > s$ (read "r dominates s") means that ownership of the right r is sufficient authority for granting a node the right s.

Warshall's note presents five rules governing the assignment, modification and removal of permissions. For this note, the most

27

important rules are the following two .

Rule 1: Grant a permission.

Node N_a may grant the permission

$$p = N_t, \text{right}(p), \text{ancestor}(p)$$

to node N_t if and only if:

26

We deviate slightly from Warshall's formulation. He includes $\text{time}(p)$, the time at which p was created, in the permission tuple. We omit it because it is not central to our discussion.

27

Two of Warshall's other three rules are of interest. One is concerned with changing the ancestor of a permission, and the other with changing the node of a permission.

1. $\text{node}(\text{ancestor}(p)) = \text{Na}$; i.e., Node Na holds the ancestor or p
2. $\text{right}(\text{ancestor}(p)) > \text{right}(p)$; i.e., ownership of the ancestor of p is sufficient authority to grant p.

It is possible to trace a given permission by means of the ancestor components back to a permission that has the null ancestor. That permission is called a "root" permission for those permissions which are derived from it. A root permission and its descendents form a tree.

Rule 2: Delete a permission.

Node Na may delete the permission

$$p = \text{Nt}, \text{right}(p), \text{ancestor}(p)$$

held by node Nt if and only if $\text{node}(\text{ancestor}(p)) = \text{Na}$.

A side effect of deleting p is that the set of permissions rooted at p are also deleted. We shall call this set T(p). T(p) is defined by:

1. p is in T(p)
2. if $\text{ancestor}(q)$ is in T(p), then q is in T(p).

The motivation for deleting descendents of p (i.e., permissions granted directly or indirectly because of p's existence) is that such permissions lose their validity when p is deleted.

10.2 Nodes and Permissions for NSW

The permissions known to the NSW system are maintained in the Node Permission Data Base. The Node Permission Data Base contains a record for each node. The node record lists the permissions held by the node. Each node record represents some user, and the node's permissions define the actions the user has been authorized to take within the system.

Development of a permission system for NSW involves two things.

1. A set of rights for NSW and dominance relations for them must be defined.
2. The two basic rules that govern manipulation of permissions need to be refined somewhat, given the rights that are defined for NSW. We shall discuss why this is necessary and present the refinements below.

Sections 6, 7, 8, and 9 have introduced permissions related to files, devices, services, and workspaces in an informal way. We now consider rights related to nodes. Rights related to files, devices, services, and workspaces will be reconsidered later in this section.

The right of a user to be known to the system is governed by a permission called an "exist" permission. That is, to have a record in the Node Permission Data Base a node must hold an exist permission.

Exist permissions may be granted by nodes that hold the permission that enable them to create and delete node records. That permission is called a "create/delete" (cd) permission. Typically cd permissions will be exercised by users acting in an administrative capacity. Figure 10-1 illustrates the relation between exist and cd permissions.

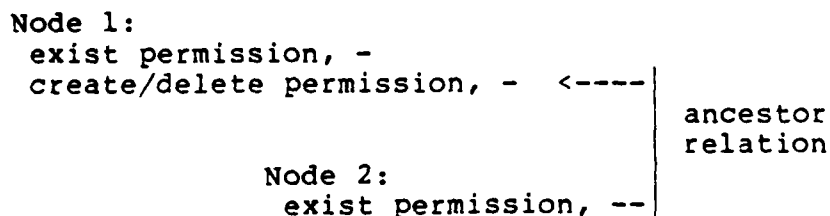


Figure 10-1: Relation between exist and cd permissions.

The following considerations have influenced our design of NSW permissions relating to nodes.

1. There is a need to support both hierarchical and non-hierarchical node structures.

This consideration recognizes the difference between project structures or organizational entities and the allocation of people to such structures. NSW must deal both with organizational entities and with people. From the point of view of NSW, an organization or project has a single reason for existence or being known to the system. A person may have multiple reasons for being known to the system.

To deal with this fundamental difference between people and organizations, we believe NSW should support at least 2 kinds of nodes, organization nodes and person

nodes.

Organizational nodes, called o-nodes for short, are intended to represent organizations. For organizations there is usually a strict hierarchy of responsibility. To model this we allow an o-node to hold only a single "organization exist" (oe) permission of the form:

$p = \text{o-node}, \text{oe}, \text{ancestor}(p).$

An o-node's oe permission is granted to it by the node which is (by definition) superior to it in the organizational hierarchy; that node is $\text{node}(\text{ancestor}(p))$ and it must hold a cd permission.

Person nodes, called p-nodes for short, are intended to model the fact that a single person may function in any of several capacities. That is, a person may belong to several organizations or participate in several projects. To model this, we allow a person record in the Node Permission Data Base to have more than a single "person exist" (pe) permission. A p-node may have one or more permissions of the form:

$p = \text{p-node}, \text{pe}, \text{ancestor}(p)$

Each such pe permission held by a person node is granted to it by another node, $\text{node}(\text{ancestor}(p))$, that holds a cd permission.

An interesting design issue is whether a single node should be able to hold both pe and oe permissions. There appears to be no reason to prohibit this on theoretical grounds. However, it is prohibited in order to preserve the distinctions between people (persons) and organizational entities.

A node has the right to exist in the Node Permission Data Base as long as it holds an exist (oe or pe) permission. An o-node may have only a single exist (pe) permission and a p-node may have several. Only when all exist rights for a p-node have been removed does the node lose its right to exist in the data base.

A given person may be granted pe permission several times by different administrators. When a pe permission is granted, it is important that it be granted to the "correct person". That is, when the pe permission is granted, the NSW core software responsible for the Node Permission Data Base must check whether there already is a node record in the data base for the user. If there is, a new pe

permission should be assigned to the node record. Otherwise, a new p-node record should be added to the Node Permission Data Base for the user, and a pe permission assigned to it. Determination of prior existence may be a tricky operation. It is discussed later in this section in the context of node names.

2. Managerial responsibilities derive from organizational entities.

The implication of this consideration is that only nodes representing organizational entities (o-nodes) may exercise "managerial" rights. Managerial rights include the right to create and delete other nodes (cd), to allocate organizational resources, to delegate responsibilities, etc.

3. Persons may act as organizational entities in order to exercise managerial responsibilities.

The implication of this consideration is that NSW should support an "act as organizational entity" (ao) permission. This would enable a person node which holds it to exercise the permissions held by a particular organizational node.

4. Ultimately persons are accountable for the actions taken within the system.

The implications of this consideration is that login to the NSW should be as a person rather than as an organizational entity. Furthermore, permissions should carry the identity of the granting person as well as the ancestral permission in order to make maintenance of an accountability record possible. That is, permissions should be enlarged to be 4-tuples, the fourth element of which is the p-node that granted the permission.

5. Any right a user can exercise derives from one of his reasons for being known to the system. Therefore, when an exist permission for a user's node is revoked, other permissions held by the node which derive from the exist permission should be deleted with it.

This observation leads to a notion of "related" permissions.

Nodes exist for a reason, the reason being why they were granted their exist permission. Because an o-node holds only a single exist permission it can be said to exist for a single reason. A p-node exists for

possibly multiple reasons. When a node is granted a permission it is given the permission in order to enable it to carry out actions governed by the permission which are related to one of its reasons for existence.

An o-node's single exist (oe) permission corresponds to the single reason for its existence. That is, all permissions it holds are related to that single reason, and therefore are related to the o-node's oe permission.

Similarly, each exist permission (pe) held by a p-node corresponds to a reason for the person node to exist. Each permission held by the person node must be related to one of the node's reasons for existence, and therefore to one of the p-node's pe permissions. In other words, a person would not be granted a permission unless it is required to carry out some action related to one of the person node's reasons for existence.

When an exist permission for a node is deleted, a reason for the node's existence is being removed. Consequently, any permissions related to the deleted exist permission should also be deleted. To permit this, we formalize the notion of related. For an o-node every permission held by the node is related to the only oe permission for the node. For p-nodes, each non-pe permission either is related to a specific pe (in which case the nodes for the ancestors of the pe permission and the non-pe permission are identical), or is related to the last remaining pe permission for the node.

To be more precise, we modify Rule 2 for deleting permissions somewhat:

First, define `related(p)` as follows:

```

if right(p) = oe, then
    related(p) = {q, such that node(q)=node(p)}

if right(p) = pe, then
    if node(p) has a single pe permission then
        related(p) = {q, such that node(q)=node(p)}
    otherwise, node(p) has multiple pe permissions and
        related(p) = {q, such that node(q)=node(p)
                       and node(ancestor(q))=
                       node(ancestor(p))}
  
```

Then, add to definition of `T(p)`:

3. if p is in $T(p)$ and $p = p_e$, then if q is in $\text{related}(p)$, q is in $T(p)$.
6. A user of the system always works within some organizational context. The user may be working as a manager, in which case the user has exercised an ao permission held by his person node that enables him to exercise permissions held by the organization node. Or, the person may be working as a staff member in which case he may exercise the rights granted to his person node by some manager.

The system supports an "act as organization" command which can be used to exercise an ao permission, and it also supports a "cease acting as organization" command.

A fundamental design decision needs to be made concerning the permissions that should be in effect at a given time for a p -node. Should all permissions held by the node in effect, or only a particular related set (i.e., the set defined by one of the node's p_e permissions)? We don't feel strongly on this issue. However, for simplicity we recommend that all the permission held by the p -node be in effect.

A related decision concerns the permissions in effect when a person is acting as an organization. Should the active permissions be the union of those held by p -node and o -node, or only the o -node's permissions? Again, we don't feel strongly on this issue. But for simplicity we recommend that the union of the permissions be used.

To summarize, so far we have two types of nodes, organizational entity nodes (o -nodes) and person nodes (p -nodes).

O -nodes have a single exist permission (o -exist or oe) and all permissions they hold are related to that exist permission. Only o -nodes may hold "managerial permissions". Managerial rights include the right to create and delete other nodes as well as others to be defined. The ancestral permission of an o -exist permission is a create/delete (cd) permission. Hence, the o -exist and cd permissions known to the system form a tree. That tree can be regarded as an organization chart.

P -nodes may hold one or more exist permissions (p -exist or pe). Each permission held by a p -node is related to exactly one of the p -node's exist permissions. System login determines the p -node for a user. A user may exercise the rights of an o -node if the user's p -node holds an ao permission for that node.

From the discussion above, it should be apparent that an o -node

in isolation is useless, since there is no way to exercise any permissions it might hold. Similarly, p-nodes can not exist in isolation, since they can only be created by exercising a cd permission held by an o-node. To minimize the occurrence of meaningless node structures in the Node and Permission Data Base, NSW core software provides some enhancements to simple node creation.

When an organization node (o-node) is created, some additional node and permission structure is also created. An associated person exist (pe) permission is created together with an act as organization (ao) permission that identifies the new o-node. In addition, the o-node is created with a create-delete (cd) permission which is the ancestor of the associated pe permission. This may (and in most cases does) result in the creation of a new person node (p-node). The exception is when a node for the person already exists, in which case the new pe permission is assigned to the existing person node. The motivation for this convention is to ensure that there is always a person node which can assume the role of the organization. Figure 10-2 illustrates the result of creating a new o-node.

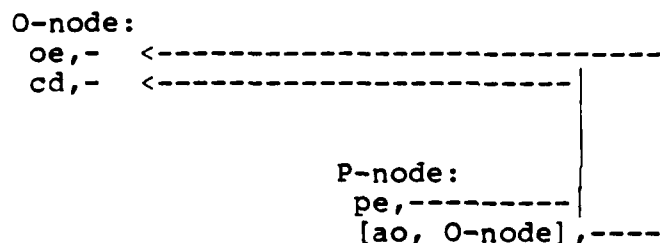


Figure 10-2:

The ancestor permissions of the associated pe and ao permissions held by the p-node are the cd and oe permissions, respectively, of the new o-node. The arrows are meant to show the ancestral relation for the permissions.

The following sequence of commands illustrates how a user might create a node.

```

NSW: login metzger pqrdef
      Richard A Metzger logged in
NSW: act-as (organization) RADCS-ISC
      Root.RADC-ISC permissions in effect
NSW: create node
      Node type: organization
      Node name: BBN-NSW
  
```

```
Name will be Root.RADC-ISCP.BBN-NSW
Own space for organization? yes
Own space name will be: BBN-NSW1
Person responsible:
Person's last name: Schantz
    first name/initial: Richard
    middle name/initial: E
No node for this person exists; create one? yes
Person's login string: Schantz
Person's password will be zxcvbn
Own space name will be Schantz1
Organization node
    Root.RADC-ISCP.BBN-NSW
and person node
    Schantz.Richard.E.1
created
```

NSW:

It might be useful to add a third type of node as an optimization of sorts and, perhaps, as a convenience. This new type of node would hold permissions that one or more users might exercise while acting in a staff member capacity. We would call this node a role node (r-node). Like o-nodes, r-nodes would have only a single exist permission the (r-exist or re). Like p-nodes, r-nodes would not be able to possess managerial permissions. To exercise the permissions held by an r-node a person node would have to hold an "assume role" (ar) permission for it.

The r-node concept is an optimization in that it allows permissions required for some task that a number of persons work on to be grouped and held by a single r-node rather than forcing the permissions to be replicated and held by all of the appropriate p-nodes. It is also a "convenience feature" in that arbitrary sets of permissions can be grouped, and their use controlled by means of a single (ar) permission. While not a direct consequence of the above considerations, the r-node concept is probably a useful one for large projects.

10.3 Node Names

The subject of node names and the lookup function for nodes is an important one. To use the functions that grant and revoke permissions and that create and delete nodes, a user must name an existing node or select a name for a new node. To login, a user must identify a p-node, and to "act as an organizational entity" or to "assume a role", a user must specify the node that represents the entity or role. Furthermore, since NSW mail (discussed in Section 12.10) is addressed to nodes, a user must name the destination node(s) in order to send mail.

A number of approaches are feasible for node naming. We recommend the following as a reasonable one.

To login a user specifies a login string and a password. Together the string and password uniquely specify the user's p-node. Users would be permitted to choose their own login strings, but not their own p-node names (see below). For example, user John A Smith might choose "Smith" as a login string. However, to ensure uniqueness of the login string/password combination and password privacy, users would not have complete freedom in selecting passwords. Passwords for users would be generated by the system. For example, when a p-node is created for a user a login string would be specified by the creator and a password would be generated by the system. The user could later, if he chose, request that the password and/or login string be changed; in such a case, the system would

28

generate any new passwords .

Each node has a unique, complete node name which is a sequence of name components. The node creation function (the entry function for nodes, see Section 5) enforces the uniqueness of complete node names. We write:

a.b.c.d

for a node name with components a, b, c, and d.

As noted above, the set of o-nodes form a strict hierarchical or tree like structure under the ancestral relation for their exist permissions. The naming conventions for o-nodes take advantage of that in that the sequence of name components in the complete name of an o-node specifies the path through the ancestral tree to the node. For example, should the o-node named p.q.r create a new o-node which it names "s", the complete name of the new node would "p.q.r.s".

28

Users may object to system generated passwords. A workable alternative might be to allow a user to pick a password, subject to system approval. To ensure uniqueness of login string/password combinations, the system would reject any password that matched that for another user with the same login string. A rejection on these grounds alone would compromise the other user's password. To prevent that, the system would also randomly reject, say 50%, of all requested passwords, whether the corresponding login string/password combination were unique or not.

The name components for a p-node are the person's last, first, and middle names, and an integer generated by the system to ensure uniqueness. For example, if there were two John A Smiths known to the system, there would be p-nodes named:

Smith.John.A.1 and
Smith.John.A.2

Each Smith would, of course, be free to choose "Smith" as a login string.

To make it easier for users to refer to nodes, the node lookup function accepts partially specified node names, called "node specs". We don't specify in detail a syntax for node specs in this note. However, we observe that system software could easily transform strings such as "Smith", "John Smith", and "Smith, John A" into the corresponding partial node names prior to invoking the node name lookup function.

A node spec may specify more than a single node, in which case the user would be required to supply more information to disambiguate it. In the case of the two John A Smiths, if the user supplied "John A Smith" as a node spec, the lookup function would find two nodes. It would then depend upon information stored with the nodes and further information supplied by the user to select the "correct" one. The manner in which the user supplies this information could depend upon the situation. For example, in the course of attempting to create a p-node for a John A Smith (i.e., grant a pe permission), the creator would be asked the name of the new user. Upon supplying "Smith", "John", and "A" for the last, first and middle names, the user would be asked if he meant the existing Smith.John.A.1 node or the existing Smith.John.A.2 node. This question would be formulated in terms of information about the two Smiths known to the system and stored in the two node records, such as mailing addresses, telephone numbers, etc. If neither of the two known Smiths was the person the user had in mind, then a new node named "Smith.John.A.3" would be created.

10.4 Own Space

Regions of NSW resource space can be allocated to users for their own use. Such a region is called an own space, and as its name suggests, the user "owns" the region.

The own space notion is supported by a right type called "own". An own permission or key specifies a region of NSW resource space "owned" by the node that holds it. The node "owns" the region in the sense that it has complete control over use of the region. Initially only a single node is granted an own key for a particular region of NSW name space. Thus, the node that holds

the own key has exclusive access to the region the key names. No other node can access the portion of file space designated by the own key unless the owning node has granted that node access to it (by giving it an appropriate key).

These properties of own spaces are ensured by the dominance relation for the own right. An own key is dominated by no other key, and it dominates any other access key which specifies a subset of the resource space it names.

A design issue concerns the way own space is initially assigned. We recommend that it be assigned to each person node (p-node) when the node is created. Thus, each p-node is created with an own key that designates a corresponding own space. The user is then free to grant other users access to his own space as he sees fit. We recommend the creator of an o-node should have the option of granting the new node an own key when the node is created.

Another design issue concerns the notion of "relatedness" for own keys held by p-nodes with multiple pe permissions. There are really two questions here. The first is: should such a p-node be able to hold multiple own keys? While we see no reason to prohibit this on theoretical grounds, we also see no compelling reason to allow it. Therefore, we prohibit it for simplicity. The second question is: to which pe permission should a p-node's single own key be related? Since the notion of relatedness is important primarily when exist permissions are removed, we recommend as an operational rule that a p-node's own key be related to the pe permission removed from the node last.

Own spaces have names. Recall from Section 5 that regions of NSW resource space are named by sequences of name components, and that the name of every object in a region named X begins with the prefix substring X. An important property of own spaces is that they are non-intersecting. Thus, no own space name may be a prefix substring of another own space name. To simplify the task of software that must ensure this property, we require that all own space names have a single component.

The names for own spaces are derived from the corresponding node names. For a p-node the name for the own space is the last name component of the node name followed by an integer. For example:

```
$Thomas1*  
$Smith2*
```

The purpose of the integer suffix is to ensure uniqueness of own space names. Similarly, for an o-node, the name for the own space is the last component of the node name followed by an integer. For example:

\$BBN-NSW1*

10.5 Permission Types

Previous sections have identified many different types of permissions. This section summarizes them.

o Node Permissions

Exist permissions govern a node's right to have a node record in the Node Permission Data Base. Two types of exist permissions have been identified, and a node may hold only one type of exist permission. The exist permissions are:

person exist (pe)

A node can hold one or more pe permissions, in which case it is a p-node.

organization exist(oe)

A node can hold a single oe permission, in which case it is an o-node.

The create-delete (cd) permission enables a node to create and delete other nodes, by granting exist permissions. Only o-nodes may hold cd permissions.

The act as organization (ao) permission identifies an o-node and enables a user that holds it to exercise the permissions held by the o-node. The notation

[ao, o-node]

is used to represent an ao permission. Only p-nodes may hold ao permissions.

o Resource Space Permissions

Resource space permissions are called keys. A key identifies a region of NSW resource space and the type of access permitted by the key to the region. The notation

[atype, xxx]

is used to represent a key that permits access of type "atype" to the region named "xxx". Resource space permissions may be held by any node type. The following keys are defined:

[own, xxx]	This is the ownership right for the specified region of resource space. Own rights dominate all other resource space rights.
[copy, xxx]	Permits read access to files in region xxx.
[enter, xxx]	Permits objects to be entered into region xxx.
[delete, xxx]	Permits removal of objects in region xxx.
[update, xxx]	Permits modification of files in region xxx.
[use, xxx]	Permits services in region xxx to be used.
[activate, xxx]	Permits access to workspaces in region xxx.

o Workspace Permissions

These permissions control the assignment of long term workspaces. There are two right types. One permits a user to have a permanent workspace on a particular host. The other permits a user to grant a permission of the first type.

workspace [ws, host]

This permits the holder to have a permanent workspace on the specified host. The workspace is obtained by means of a user level assign workspace command.

assign workspace (aw)

This permits the holder to grant ws permissions. It may be held only by o-nodes.

o Other Types of Permissions.

As previous sections have discussed, there may be a need to control the registration of various types of objects, such as devices and services. We believe the nature of such access controls is more a policy issue than a technical one. Permissions that implement a variety of policies can be defined to control their registration in a relatively straight forward manner. We refrain from

defining such permissions until a policy is established.

10.6 Node Records

Implementation of the NSW functions described in this report requires that certain information about nodes be maintained in the Node Permission Data Base. This section summarizes the information that should be maintained with the node records for p-nodes and o-nodes.

The following information is maintained for p-nodes.

Node Name

Login String

Login Password The combination of the login string and the login password must be unique across the system.

Node-id This is a unique identifier for the node that is used internally by NSW core software.

NSW core software implements a node lookup function which retrieves node records from the Node Permission Data Base, given a specification of the node. The specification supplied to the lookup function depends upon the situation in which it is invoked. At login time it retrieves a node given a login string and password. When mail is being addressed or permissions are being granted to a node, the lookup function is called with a node spec. At other times it may be called with a node-id.

Permissions Permissions held by the node should be grouped into related sets to facilitate the deletion of pe permissions. For example:

pe Set 1

own

...

pe Set 2

own

...

unrelated permissions (i.e., ones related to last remaining pe permission)

Default File Scopes

These scopes are used at login time to set the scopes initially in effect for the user's session.

Assigned and registered workspaces

These are the names of permanent workspaces owned

by the user. They are used by the NSW core software that implements the "use" command for program and WS-CI services to determine whether it is necessary to allocate a workspace.

Mail repository location

Name of preferred mail service

Node has new mail flag

O-nodes with new mail

This is a list of o-nodes for which this node hold ao permissions and which have new mail (See below and 12.10).

Uninterpreted data

This is data that is not interpreted by core NSW software and which is intended for use by application software. Although the data is not used by the core software, its format will be defined. This data could be stored in a related data base rather than in the node record itself. We expect the data will include information such as:

user name
user telephone number
user US mail address

The following information is maintained for o-nodes.

Node name

Node-id

Permissions All o-node permissions are related to the single oe permissions.

oe
cd
...

Mail repository location

Name of preferred mail service

Node has new mail flag

Nodes to notify when mail for o-node arrives.

These are nodes that hold ao permissions for this o-node. Notification is accomplished by adding

the node-id for this o-node to the "o-nodes with new mail" list in each of their records.

Uninterpreted data

10.7 Granting Permissions

This section includes an example illustrating how a user might grant permissions to another user.

Below we assume user Schantz logs in to grant user Thomas file permissions (file keys) to regions of the own space for the BBN-NSW project.

```
NSW: grant permission
Node name: Thomas
Granting to Thomas.Robert.H.1
Permission: enter
    for region: $BBN-NSW1.NSW-Documents*
Permission: copy
    for region: $BBN-NSW1.NSW-Documents*
Permission: enter, copy, delete, update
    for region: $BBN-NSW1.NSW-sources.MSG*
Permission: enter, copy
    for region: $BBN-NSW1.NSW-executables*
Permission: enter, use
    for region: $BBN-NSW1.Programs*
Permission:
The following permissions:
    [enter, $BBN-NSW1.NSW-Documents*]
    [copy, $BBN-NSW1.NSW-Documents*]
    [enter, $BBN-NSW1.NSW-sources.MSG*]
    [copy,, $BBN-NSW1.NSW-sources.MSG*]
    [delete, $BBN-NSW1.NSW-sources.MSG*]
    [update, $BBN-NSW1.NSW-sources.MSG*]
    [enter, $BBN-NSW1.NSW-execuatbles*]
    [copy, $BBN-NSW1.NSW-execuatbles*]
    [enter, $BBN-NSW1.Programs*]
    [use, $BBN-NSW1.Programs*]
added to node for
    Robert H Thomas
```

Here, the permission type determines the sort of question the granter is asked. The purpose of the question is, of course, to completely specify the permission. For the convenience of the user, the system allows the user to specify a collection of file permissions at once; for example, the enter, copy, delete and update keys for \$BBN-NSW1.NSW-sources.MSG*.

10.8 Summary

Figure 10-3 illustrates the relations among the notions discussed in this section.

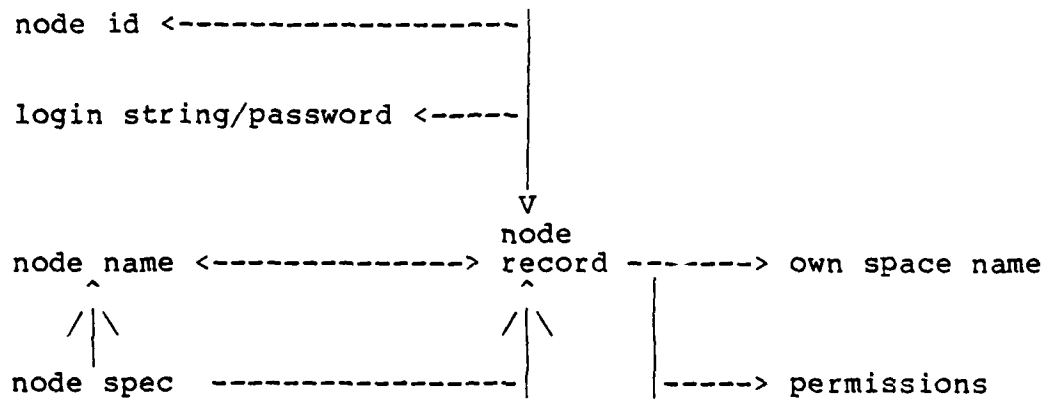


Figure 10-3:

A one way arrow (a ---> b) means that given a, b can be obtained. For example, given a node spec, the core software can obtain the corresponding node record; given a node record, the system can obtain the name of the node's own space, if any. A two way arrow (a <--> b) means that either a or b can be obtained given the other. For example, given a login string and a password, the NSW core software can obtain the corresponding node record in the Node Permission Data Base, and given the node record, the software can obtain the login string and password.

11. User Sessions

The login command creates user sessions, and the logout command terminates them.

When a user logs in, a new record for the user session is added to the Session Record Data Base. The session record defines the user's session environment. It holds information about the session that is required by the system to perform its functions. For example, it is used by the system to initiate access control checks and to support various user level status commands (see Section 12.1).

The record is initialized from information stored with the user's node record in the Node Permission Data Base, and it is modified as the result of user actions during the session.

A session record contains the following information:

NSW session-id. This is a unique identifier for the session which uniquely identifies it. It is used internally by NSW software.

The node-id for the user's p-node.

The node-id of the o-node for the organization, if any, for which the user is acting.

User access point information.

This includes the access host, the terminal-id for the user's terminal, the process-id for the user's NSW command interpreter, etc.

The permissions in effect.

These define the actions the users are authorized to perform. Initially the permissions are those held by the p-node identified at login time, but may change as the user exercises any "act as organization" (ao) (or "assume role" (ar)) permissions held by the p-node.

The scopes in effect.

These define regions of name space accessed frequently by the user. The scopes are largely a convenience feature which allow users to specify partial rather than complete names for NSW objects, such as files. The scopes are initially set from the user's p-node record at login time, and may be altered by NSW commands.

Active services.

Initially there are no active services when a

user logs in.

Rerunnable services.

These are service sessions interrupted by system or component failures prior to their normal termination, which the user can access to recover temporary results.

12. Other Aspects of the Design

12.1 Status Functions

A principle of the NSW design is that all information stored in the system about objects, user sessions, and the state of the system itself can be displayed by means of user level commands, subject to privacy considerations and access control restrictions implemented by the permission system. Similarly, subject to system integrity considerations and access control constraints, a user should be able to change much of the information maintained by the system.

The information supplied by the status functions comes from a number of sources including the Resource Catalogue, the Node Permission Data Base and the Session Data Base.

Information about NSW objects, with the exception of node objects, is stored in the Resource Catalogue. The following is the type of information a user should be able to obtain about NSW objects.

- o For file objects:

- NSW name for the file
 - Host that stores the file
 - Name the file is stored under at the host
 - Size of the file
 - Dates of creation, last read, last write

- o For device objects:

- NSW name for device
 - Device host
 - Device access mechanisms

- o For service objects:

- NSW name for the service
 - Type of service
 - Host that supports the service
 - Type dependent information

- o For workspace objects:

- NSW name for workspace
 - Host where workspace resides
 - Type of workspace
 - temporary
 - permanently assigned
 - registered external user slot

Information about nodes is maintained in the Node Permission Data Base. A user should be able to obtain the following information about a node.

- Node's full name
- Permissions held by the node
- Default scopes for the node
- Mail system parameters (See Sections 10 and 12.10)

A user should be able to obtain information about his own session as well as (perhaps more limited) information about sessions for other users. This information would come from the Session Data Base.

It should be possible to obtain information about the state of the NSW system. Information of interest includes:

- o The hosts which are part of the system configuration.
- o The hosts which are currently active in the configuration.
- o The roles these hosts play in the system (e.g., service bearing host, file bearing host, user access point).
- o The NSW capabilities these hosts support (e.g., WS-CI service, batch service, native mode file access, etc.).

Futhermore, system operations personnel should be able to use user level commands to obtain additional information about the system status such as:

- o The number of user slots managed by NSW on each of the hosts and a breakdown on how they are being used.
- o The amount of file space managed by NSW on each of the hosts.
- o The amount of file space unused, and therefore, available for new files on each host.

Most of the status functions could be made available to users by suitable extensions to the "show" command provided by NSW 4.1. The following examples are included to suggest how this could be done.

Information about an NSW object could be obtained by a "show descriptor" (to show the information in the descriptor for the object) variation of the show command. For example, to obtain information about the file MSG.Aspec the following command could be used

NSW: show descriptor file MSG.Aspec

- . information about the file including:
- . full name - \$Thomas1.MSG.Aspec
- . size
- . creation date and time
- . last read data and time
- . last write date and time
- . host - ISIE
- . name of file on host - <...>...
- . etc.

Information about other objects can be obtained in a similar fashion as the following examples illustrate. As with all commands which reference NSW objects, the system uses the user scopes in effect to search the NSW resource catalogue. The lookup function has been described in Sections 6.3, 7, 8 and 9.

NSW: show descriptor device lpt.bbnd

- . information about the device including
- . full name - \$Devices.lpt.bbnd
- . type - line printer
- . host - BBN-TENEXD
- . access mechanism - FTP to LPT:
- . etc.

NSW: show descriptor service rome-multics

- . information about the service including:
- . full name - \$Services.rome-multice
- . type - ICP
- . host - RADDC-MULTICS
- . contact socket - 1

NSW: show descriptor workspace ie,r20

- . information about both workspaces including:
- . full name - \$Thomas1.ie
- . type - registered external
- . host - ISIE
- . user slot - BTHOMAS
- . full name - \$Thomas1.r20
- . type - internal assigned
- . host - radc-20
- . user slot - nsw-ws5
- . etc.

Information about a node could be obtained by means of a "show node" variation of the show command.

```
NSW: show node Thomas
      node name: Thomas.Robert.H
      created by: Root.RADC-ISCP.BBN-NSW
                  Root.RADC-ISCP.BBN-NSW-FE
      mail repository: BTHOMAS@ISIE
      mail workspace: Thomasl.ie
      mail service: Public.Services.Hermes
      permissions:
        [own, $Thomasl*]
      .
      . etc
      .
```

Information about a session could be obtained by a "show session" variation of the show command.

```
NSW: show session
      user node: Thomas.Robert.H
      acting as: Root.RADC-ISCP.BBN-NSW-FE
      active services:
        instance type
        name
        bbnd      ICP service
        srccom     program service
        r20        WS-CI service
      scopes:
        copy: ...
        update: ...
        enter: ...
        use: ...
        device: ...
        .
        .
        .
```

Finally, information about logged in users could be obtained by a "show system users" variation of the show command.

```
NSW: show system users
```

Session	User	FE host
1	Metzger	RADC-20
2	Lind	BBN-UNIX
4	Thomas	BBN-UNIX
5	Jones.Steven	ISIE
.	.	.
.	etc	.
.	.	.

12.2 File Placement

User control of file placement should be possible. That is, a user should be able to instruct the system to store a particular file on a particular host. There should be a command at the user interface for doing this. To be effective, this command should be used in conjunction with the "show descriptor file ..." command to discover where the file is stored (See Section 12.1).

The system will retain multiple images (copies) of moved files. However, it will make no attempt to update all images if the file is modified. Instead, when a file that has multiple images is updated, only the new modified file copy is retained, and all previous images of the file are discarded.

12.3 Importing and Exporting Files

There should be convenient means to move files between the NSW file system and the file systems of ARPANET hosts.

A limited capability for file movement of this sort is provided by the import and export functions as implemented in NSW 4.1 and by the workspace notions described in this note. Import moves a

29

file from the file system of a host that is an NSW file host into NSW file space, and export moves one from NSW file space into the file system of an NSW file host. The new workspace notions provide similar capabilities.

The limitation is that files cannot be moved between NSW and hosts which are not NSW file hosts. The limitation should be removed by extending the domain of the import/export operations to include all network hosts.

To do this, there must be a way to move files between NSW file hosts and non-NSW file hosts. The technical problem here is that the file movement mechanisms used by NSW are incompatible with the standard file transfer protocol (FTP) used by network hosts. This problem can be solved by adding to the NSW core software the ability to use FTP when necessary to transport files to and from non-NSW hosts. For example, to import a file from a non-NSW file

29

An NSW file host is one that stores NSW files and therefore implements the NSW file movement protocol; that is, a host that has an NSW File Package module (see Section 15.1). For purposes of this discussion, a non-NSW file host is one that does not implement the NSW file movement protocols.

host the core software could use the NSW FTP module to move the file to an NSW file host where it could then be entered into NSW file space.

12.4 File structure, representation, translation

12.4.1 Representational Issues

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

The NSW design objective here is to provide system level support for a variety of applications, but not to directly support every application with a set of application dependent file types that the system knows how to handle. That is, the system should not be expected to know about data representations used by various applications. In practice, the file structures some OS's support have been dictated by the applications and as a result, the systems know a lot about application data structures. That is, the boundary between system file representation and application file representations is, in practice, likely to be defined operationally rather than rationally.

12.4.2 File Movement from Host to Host

When a file is moved from one host to another, two different things are transmitted: descriptive information about the file, and the file data. Each is transmitted as a formatted stream of bits. A set of NSW standard transmission formats are defined.

The descriptive information is transmitted to enable the file to be reconstructed in an information lossless fashion. This

information may be host dependent, in the sense that not all receiving hosts may be able to interpret or make use of it. However, every host is assumed capable of storing the descriptive information supplied by any other host in an information lossless manner.

The file data is transmitted in one of several NSW standard data transmission formats. Two likely formats are "stream" for sequential files and "blocked" for record structure files. Again, every file host is expected to be able to store file data it receives in an information lossless manner, even if that means that the host must store the stream of bits that represents the file as they were transmitted without doing any unblocking, etc. Of course, the receiving host must be able to transform the data from the transmitted format to a locally supported file structure if locally supported services are to use the file in a meaningful way without performing the transformations themselves.

12.4.3 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 host. The system will support and perform a standard set of file translations both between different data representations and between different file structural types. In addition, it will be possible to define application programs (services) which perform special purpose translations.

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	

A design issue that requires further thought is the manner in which file translations are invoked. In NSW 4.1 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 filing system at the host where the file will be used, and the nature of the service that will use the file. While the system should continue to support automatic translations of this sort, users should be able to override it by specifying translations they want. One way to do this would be to permit users to modify file specs (for a file used as a data source) with the specification of a transformation. Where a file spec is expected in a user level command or a system call, the spec could specify the translation desired. For example, the command

```
NSW: copy a.b(record-text->sequential-text) r20!c.d
```

would copy the specified NSW file to the workspace named r20 by first translating the file from record structured text to sequential text.

The system would support a set of basic translations, which users can invoke. In addition, users should be able to define their own translations which they may also invoke by name in the same way as the system supported ones. That is, by naming the translation as a part of the file spec; the name would identify what is, in effect, a user program that performs the translation. To support user specified translations a system-wide set of conventions which govern programmed translations must be defined.

12.5 File Semaphores

Users need to be able to "lock" files in order to control access to them as they undergo modification. The idea is that locks would prevent other users, who are authorized by permissions to access the files, from accessing them while they undergo change.

NSW 4.1 implements file "semaphores" as a means to provide a locking function. However, the NSW 4.1 approach to semaphores is not well thought out. Below we suggest an approach to file locking.

There seem to be several different ways that a user may want to lock a file:

1. Exclusive access. Here the system would guarantee that only the user who set the exclusive access lock could access the file. No one else could access the file in any way. All attempts by other users to access a file with an exclusive access lock set would fail.
2. Write lock. Here the system would ensure that only the user could modify the file. Attempts by other users to write the file will fail. In addition, attempts by

other users to access the file in other modes with the write lock set will fail, unless the attempts explicitly specify that the write lock is to be ignored. This permits, other users to access the file in other modes while it is locked only if they explicitly state they are willing to do so.

3. Warning tag. This is the weakest type of lock. Here the system would only warn other accessors that the file was tagged. When a user has set the warning tag for a file, attempts by other users to access the file will fail unless they explicitly specify that the warning tag should be ignored. Accessers willing to ignore the tag may access the file as they please. However, it is expected that "protocols" between cooperating groups of users will develop where the warning tag is honored.

Another aspect of the lock notion is the duration that the lock remains set. There seem to be two useful durations:

1. Indefinitely. In this case the lock remains set until explicitly cleared by the user that set it.
2. For the current session. Here the lock remains set for the duration of the session that the lock was set (unless it is explicitly cleared first). When the session ends, the lock is automatically cleared. In the event a crash interrupts a session, when the user next logs in, he will have the option of clearing the lock, or having it remain set for the duration of the new session.

Each file will have a lock which may be in one of four states:

free
exclusive access locked
write locked
warning locked

In addition, if set, the lock will carry with it the id of the p-node that set it, and the duration of the setting, which is either indefinite (i.e., until cleared) or for the duration of the session.

User level commands are provided to permit users to manipulate locks. A user can lock a file (i.e., place its lock into a non free state) only if the lock is clear (i.e., in the free state). After a user has set a lock into one of the locked states, he is free to change the lock state or the duration of the lock setting.

The user's record in the Session Record Data Base will hold a list of the files locked for the duration of the session. The user's p-node will carry a list of the files locked for an indefinite duration.

12.6 File Version Numbers

Many modern operating system support version numbering for files. The idea is that different versions of a file are logically related in that they are different refinements or improvements of the "same" file. To emphasize the relation of the versions, the different versions share the same name and differ only in version "number".

Version numbering appears basic to supporting software that implements configuration control disciplines. For this reason, we believe that NSW should provide a version numbering capability for files.

It is difficult for the system to enforce the "relatedness" of versions of a file since their relation to one another is a property that is largely in the mind of the user. The system can, however, provide features that make it easy for users to group different versions of the same file.

The notation

a/1, a/2, a/17

is used to denote versions 1, 2, and 17 of file a.

The model for file objects developed in Section 6 is extended to include a version number space for file names. This is done through the catalogue entries for the names. That is, different versions of the same file are recorded in the CE corresponding to the file name (See Figure 12-1). Notice that this implies that versions are a "property" of the CE for the file rather than of the file i-node.

There is a design issue regarding the selection of a value for the file version number when a user doesn't identify a specific version. Experience with similar facilities in other operating systems indicates that the appropriate default version number depends upon the operation to be performed on the file, and furthermore, that different users frequently have different opinions regarding which default is appropriate.

This report doesn't specify any system-wide version number default conventions. However, we make two observations:

- o At the lowest level, the file system operations should

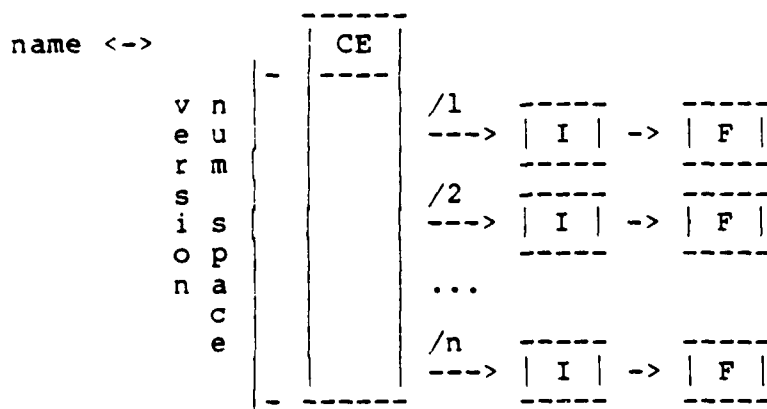


Figure 12-1:

always require a specific version number. Higher level software "nearer" the user should implement the default conventions.

- o Regardless of the system-wide default conventions chosen, it is highly likely that some users will consider them inappropriate. User specific and user settable default conventions could be provided through a user profile capability.

Since link CEs identify i-nodes rather than other CEs and since versions are bound to CEs rather than i-nodes, situations can occur where a link, which when created identified the i-node corresponding to the most recent version of a file, no longer identifies the i-node of the most recent version. For example, suppose b is a link to a/3, and then suppose a/4 is created. What should happen when a user attempts to reference the file using the name b; should he get a/3 or a/4? As with selection of default version numbers, there is probably no answer to this question that is correct for all situations. Before formulating an approach, it is useful to reconsider how the version number and linking features are most likely to be used. We expect that version numbers will be most commonly used by the creator and/or owner of a file as the file is refined and improved (i.e., by the person "responsible" for the file). Thus, we expect that OCEs for files are likely to define more versions than CEs which are not OCEs. The linking feature is intended to facilitate sharing, and link CEs are unlikely to define more than a single version (although they are not explicitly prohibited from doing so). The

creator of a link is likely to be more interested in the "most recent" or the "most stable" or the "least buggy" version of a file than in some specific version (number) of the file. Thus the link creator and user are likely to be interested in changes in the version number space of the OCE of the link target.

Since the i-node which is the target of the link identifies the OCE, the file system can determine whether more recent versions of the file exist when it is referenced through the link. The link CE could include the action to be taken when the associated i-node no longer corresponds to the highest file version. Part of the link creation (entry) operation would be to specify the action (either explicitly or implicitly by default). Possible actions include:

- o select the version associated with the i-node.
- o select the highest version.
- o select the highest version, and redirect the link CE to identify the i-node associated with the highest version.
- o ask the user attempting the file operation what to do.

This note doesn't pursue this aspect of the file system design further other than to remark that the system design appears to provide adequate flexibility.

Since versions are likely to be made mostly by file owners and links are likely to be used mostly by file sharers, it may be a reasonable design decision to allow only owning catalogue entries (OCEs) to support versions, and to prohibit versions for link CEs. If this were to be done, the version number space of the OCE could be used with file references made through the link CEs. We don't feel strongly on this issue at the present time.

Another issue is the manner in which versions should be supported for set files. We allow

a/3!b

which denotes member b of the third version of the set file a. Should the system support versions of members of set files? That is, should we allow

a!b/3 and a/3!b/4

which would denote version 3 of the b member of a, and version 4 of the b member of the third version of a, respectively? To do so would require that the set definition file (that is, the set definition part of the set file) be expanded to support version numbers for member files. Our inclination at this time is to

disallow versions of members of set files until users have had some experience with the version number and set file features, and their interactions.

12.7 File Attributes and Tags

NSW allows certain properties, called attributes, to be associated with objects catalogued in the Resource Catalogue.

The system records these attributes, and for certain attributes may restrict the types of operations that can be performed on objects bearing them. In terms of the resource catalogue model developed in Section 5, object attributes are recorded in the object descriptor in the object i-node.

This section considers attributes for file objects. There are two types of attributes. The first are pre-defined and integral to the operation of the system. The system understands these attributes to the extent that it provides operations for dealing with objects that have them. All file objects have the "file" attribute which specifies their object type. Two predefined file attributes have already been discussed. These are the NSW set file and local host set file attributes.

The second type of file attributes are user and program definable. These are called "file tags". They are intended to allow files to be "stamped" that they meet certain criteria that are of interest to users and programs, such as "syntactically correct Fortran source code" or "optimized PL/1 object code". The system itself does not "understand" these attributes but merely records them as a service to users.

In addition to the set file attributes just mentioned, the pre-defined attributes include:

- data representation type
- file structure type

- timestamp of creation
 - " " last read
 - " " last write

node that created file
" " last read
" " last wrote

size of file (units?)

manner of creation (copied from NSW file, imported from external,
imported from workspace, created by service)

source (depending upon the manner of creation: name of copied
file, name of external host name of WS, name of
service)

A user or a service can obtain the values of these attributes
for a given file from the NSW either by means of a user level
command ("show descriptor file") or a system call.

Users can associate tags with files. A tag is an tuple of the
form:

<node, how tag created, tag data>

where "node" is that node that created the tag, "how tag created"
specifies either "user command" or the name of a service, and
"tag data" is the actual tag. We expect that "tag data" will be
a short string. The tag data will be uninterpreted by NSW core
system software, but will be able to be read by programs and
users.

As with the pre-defined file attributes, users and services can
set and obtain the tags associated with a file by means of user
level commands and system calls.

We expect attributes and tags will be typically used in two
ways.

1. As modifying "filters" for file specs. The intent of
the filter would be to modify the behavior of the basic
NSW name lookup function (see Section 6.3) to return
only that file (those files) which matches the file
spec and has the specified attributes. Additional
design work is required to define a syntax for the
specification of file specs that include filters made
up of tags and attributes.
2. As a means for a program to check that an input file
has certain properties, as recorded by its attributes
and tags, that are required by the program.

12.8 Additional Levels of file storage

Additional levels of file storage and support for files can be imagined. For example,

- o Archival service to move infrequently accessed files off-line;
- o Periodic backup of the file system for reliability;
- o Access to the data computer.

These are all important system features. However, we believe that they are not important in the short term. Furthermore, the system design documented in this note can be extended in a straight forward fashion to accommodate such capabilities.

12.9 Support for Configuration Control

The design principle here is to provide features in the system which can be used by programs to implement configuration management schemes. The system will not by itself perform configuration management. Rather, it will only provide a set of basic functions which can be used by specially developed configuration management tools.

Some of the functions already described have been motivated largely by what we understand to be requirements of configuration management programs. These functions include:

- o Version numbers for files.
- o Set files for aggregating related files.
- o File descriptor information which specifies the origin of the file name of the service that created the file, or indicates that file was produced by a copy, import, etc. operation.

The following features should be added to this set:

- o Ability to catalogue "procedures" that generate new systems and perform other configuration management functions. A catalogued procedure would be an NSW file which would be interpreted by the NSW command interpreter.
- o File keys (permissions) that can restrict access to regions of file space to certain services. For example, the key,

[enter, \$BBN-NSW1.Release-Packages*], Release-tool

would allow the node holding it to create files in the specified region of filespace only by means of the specified service, "Release-tool".

From the above basic functions a variety of powerful tools for software configuration management could be constructed.

12.10 Mail

Electronic mail services would be provided by one or more mail tools accessible as program services. These tools would be used to compose and send messages, and to read and process received messages.

It is desirable that the system support the exchange of mail between ARPANET users and NSW users, as well as intra-NSW mail.

We assume that ARPANET mail is addressed to user slots that are external to NSW. That is, an ARPANET mail address is of the form "name at host" and we assume that "name" identifies a user slot on the specified host.

Mail may be sent to any NSW node.

An NSW mail address is a node spec (see Section 10) which identifies the node to receive the mail.

The node specification in the mail address is used during the mail delivery process to determine the node to receive the mail and the means by which the mail should be delivered to the mail repository for the node. The steps involved are roughly as follows:

1. The node specification is used to find the desired node, and the node record is retrieved from the Node Permission Data Base. If there is no node that satisfies the node spec or if the node spec is ambiguous, an error indication is returned.
2. The location of the node's mail repository is obtained from the node record.

We allow three cases for the mail repository:

- a. Normal ARPANET repository. Here the repository is identified in the standard way; i.e., "name at host"
- b. Normal ARPANET repository (name at host) plus a

registered (or assigned) NSW workspace that corresponds to (or holds) the repository. The additional information here specifies the workspace to be used when a mail tool is instantiated for the user. The idea is that the host user slot that holds the mail repository is the same user slot that supports the NSW workspace named.

- c. NSW file as a repository. Here the mail repository is an NSW file. Although there is no absolute requirement to limit the names for such files, we shall do so. By convention, the name of the file for a node's mail repository will be:

\$own-space-name.mail

NSW mail files will have a special "mail file" attribute. The system uses this attribute to ensure the integrity of the mail file by allowing only mail service software write access to files that bear it.

The mail repository information stored in the node records, in effect, defines a mail forwarding data base not unlike the one in use for several years at the BBN Research Computer Center.

3. Finally, the mail is delivered to the node mail repository and the "you have new mail" flag in the node
30
is set . In addition an indication is set in any other nodes which have "act as organization" (ao) (or "assume role" (ar)) permissions for the node.

The method of delivery used depends upon the type of mail repository used by the node. For those that are normal ARPANET mail repositories, the standard ARPANET delivery mechanisms are used. For a repository that is an NSW file the mail will be

30

It may be tricky to set the mail flag at the correct point in time, especially if delivery of the mail is by normal ARPANET means which operate asynchronously with respect to NSW.

31

delivered to the repository using NSW file movement mechanisms .

When a user logs in, the user will be notified if there is any new mail of interest. In addition, a user may query the system during a user session whether any new mail of interest has arrived. "Mail of interest" is defined to be mail addressed to the user's p-node or to any of the o-nodes for which the user holds an "act as organization entity" (ao) permission.

The NSW strategy for running mail services for a user is to run the mail program at the host that stores the user's mail repository. We consider each of the above three cases:

1. For a normal ARPANET mail repository, the system has no information about which NSW workspace, if any, is bound to the user slot that supports the mail repository. Therefore, the user must use an ICP service to access the host. NSW can give this user little help beyond providing a convenient way to access his mail host. The user must login to the host and invoke the mail service himself.
2. In the case of a normal NSW repository with a companion NSW workspace, the user slot that holds the node's mail repository is also the user slot that is used to support the workspace stored in the node. In this case the user's node holds the id of the companion workspace. Here the system would start the mail service program in the companion workspace.
3. All NSW files that are mail repositories could be stored on the same NSW host, or at least on hosts of the same type. To run the mail service for a user, NSW would allocate a workspace on the host storing the mail file and start up the mail service in it. NSW software on the host would ensure that the mail tool had direct access to the user's NSW mail file.

As the above discussion suggests, to properly provide access to a user's mail repository the system must consult the user's node to determine which of three possible cases apply.

Thus, the mail services must be treated differently from other

31

Alternatively, this could involve the use of standard ARPANET mail transport mechanisms to move the mail to an NSW system mail box at the mail file host, followed by redistribution to the mail file by NSW software.

services since data in the user's node record, specific to the service, determines how the service should be provided. The system must, of course, know to consult the node record for this information when the mail service is requested by a user. The system could be signalled to do this in either of two ways:

1. A special mail command for invoking mail services could be provided. This would signal the system to consult the node record prior to instantiating a mail program.
2. The service object itself could hold an attribute that identifies it as a mail service. In this case the user would invoke the service in the normal fashion, and the system would detect the "mail service" attribute and know to consult the node for the necessary information.

At present we prefer the first approach. The user node will contain information that specifies the mail repository, one of the three access mechanisms (i.e., ICP, activate specified permanent workspace, activate temporary workspace with access to NSW mail file), and the (name for the) user's preferred mail program. Using this information the system can properly instantiate the mail service for the user.

When a user sends a message, the addressing conventions in effect depend upon the manner in which the mail tool was being run. The same three cases discussed above apply.

1. Here the mail tool runs completely externally to NSW in a native mode, and the standard ARPANET conventions apply. Getting mail into NSW may be a problem. What is required is a way to specify that the mail is for NSW and what the node spec is. An address of the form

node-spec@nsw

would be adequate assuming the host or the mail tool recognized "nsw" as a pseudo-host.

2. Here the mail tool is running in an NSW workspace that the user has registered or has had assigned. We require that hosts that provide mail services in workspaces support an NSW-wide standard for mail addresses. We propose a simple standard that uses

node-spec@nsw or node-spec

for NSW addresses and

name@host

for ARPANET addresses. A host could at its option

provide address validation for NSW addresses (i.e., checking at address specification whether the node-spec actually unambiguously identifies a node.

3. Here again the mail tool is running in an NSW workspace, one that has been temporarily allocated to the user. As in the previous case the mail tool/host must support the NSW-wide standard for mail addresses.

12.11 User-User Interactions

NSW mail services provide means for non-real time user-user interactions.

There are situations where on-line interactions, such as those supported by "terminal linking" on some hosts, are desirable. For example, a user might want to ask a system operator a question, an operator might want to notify a particular user of some situation of interest to the user or to notify all logged in users of some change in the system status or its configuration, or one user might want to carry on a conversation with another.

The system will support a mechanism for on-line interaction of the following sort.

A user may initiate an interaction with another user by means of a command that specifies the user he wishes to interact with. For example, he might identify the target user by a node spec or by the session number obtained from a "show system users" command (see Section 12.1). The system notifies the target user, informing him that the initiating user wishes to interact with him, and asks whether he is willing to engage in such an interaction. If the target user agrees, the system establishes a communication path to support interaction between the users. If not, the initiating user is notified that the attempt to initiate the interaction failed.

After a path is established the two users may interact by exchanging "messages". A command for transmitting a body of text (typically user typein) as a message over the path is provided by the system. The command accepts a message from one user and transmits it to the other user, using the established communication path. At any time either user may terminate the interaction.

It is conceivable that a user may be involved in interactions with more than one user at any given time. To support such interactions, the system assigns a name to each interaction (or the communication path that supports it) so that a user can indicate to which conversation his various actions, such as those that send a message or terminate an interaction, are meant to

apply.

The system provides suitable means for a user to indicate his desire to refuse all attempts by other users to initiate interactions, to accept all attempts automatically, to control when messages received from other users are printed on his terminal, and so forth.

In addition to the user-to-user interactions just described, the system will provide means for a suitably authorized user, for example an operator, to broadcast a message to other users, such as all logged in users or some specified subset of them. Means will be provided to enable a user to inhibit such messages from being printed on his terminal, either by having the message discarded or its printing temporarily delayed.

12.12 Information services

The system should provide a wide range of user help and assistance facilities. These would range from simple help features typically found in command language interpreters, such as prompting and simple query support (e.g., question mark), to on-line documentation of the system and services, to sophisticated informational data bases, such as the ARPANET NIC provides.

The system design principle here is to provide a basic set of features that will support these informational services. We believe that the design documented here does so. Elaboration of the information services to be provided is beyond the scope of this document

13. Operator Functions

The approach to supporting the operation of the NSW was outlined in the "NSW-CONCEPT" document. A detailed discussion of operator functions will be the subject of another document.

14. Summary of User Model

The previous sections introduced the basic concepts associated with the NSW file system, workspaces and tool services. This section summarizes and relates concepts from the perspective of the NSW user.

Recall that there are two distinct, but related storage systems for supporting the file storage requirements of NSW users and services. The NSW file system provides 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 to support all of the expensive distributed file system attributes for all file objects. Thus, users are expected to understand the differences between files managed by NSW core software and those managed by NSW service/user interface software.

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.

14.1 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. (See Section 6.3)) 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. Users may request that multiple images of a single NSW file be maintained by the system and used interchangeably to satisfy references to the file.
7. The NSW core software may choose to move an NSW file or an image of an NSW file from one host to another as required to support remote access file references, provided the transfer is information lossless and not explicitly prohibited by user command.

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.

The NSW workspace concept was developed largely as a means to interface a diverse set of existing programs and services to the NSW system, and to reduce the high overhead associated with accessing the NSW file system.

Files in workspace storage areas are managed by NSW host software that also acts to provide an interface between the service's local host environment and the NSW system environment. The service workspace represents a boundary beyond which user-level uniformity is not required except in the few areas noted in Sections 8 and 9. This non-uniformity directly reflects the varying capabilities of the hosts participating in the NSW

system, and the diversity is, in some, ways a positive attribute of a heterogeneous network system. Workspace files are not part of the NSW file system, although there exist NSW system mechanisms for conveniently integrating workspace file activity with NSW file system activity.

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 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.

NSW supports a number of features that facilitate the use of workspace file storage facilities with NSW file storage facilities. One of these is the ability to catalogue the workspace as a named object in the NSW resource space. Coupled with host set files and syntactic conventions which allow users to control the domain of their file references, users can develop

a style of use whereby, conceptually at least, the workspace represents a unique region of NSW object space. Files in this "region" are, however, physically bound to the workspace host and do not conform to uniform NSW system syntactic and semantic conventions. Subject to these limitations workspace file objects may be manipulated from outside the context of the workspace and NSW file objects may be manipulated from within the workspace in a uniform manner.

14.2 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. import). 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 from (^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.

14.3 Types of Services

In pursuing the design goal of providing access to as wide a class of different services as possible, the NSW system supports mechanisms for starting various forms of services.

For ICP services, NSW provides no user workspace, and there is no integration of the ICP service with the NSW file system.

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 or NSW command interpreter 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 or the NSW command interpreter. 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 the 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. Some NSW mode services may utilize NSW conventions for both workspace and NSW file operations, whereas others may utilize service or host specific conventions for workspace files.

14.4 Summary of Naming Conventions

All retained NSW objects are named with a common syntax which consists of an ordered sequence of name components. The name can be thought of as a pathname through NSW name space. Objects can have multiple names as outlined in Section 6. Regions of NSW name space also are named with an ordered sequence of name components, with the objects of the region being those objects whose names begin with the ordered sequence of name components that name the region. Users obtain access rights to regions of the object name space through the permission system.

Part of the user session data base records the regions of name space which are used as scopes for the NSW object lookup and entry operations. Scopes serve to limit the area of name space searched during lookup operations and to define the area of name space used for entering new objects. Scopes relieve a user from having to write all components of a name when specifying objects, and allow the system to focus on an area of name space which the user has indicated he is likely to be use. Scopes can be applied in series whereby a set of different regions are searched sequentially in an order previously specified by the user until a matching object is located, or in parallel, whereby all regions in the set in effect are searched and all matching objects reported to the user for any further disambiguation. The symbol "\$" is used to inhibit the scoping mechanism when specifying an object spec, and allows users to provide unscoped full object names. When entering an object, the entry function uses the spec supplied by the user along with the scopes in effect to generate a complete name for the entered object (see Section 6.3).

Set files (see Section 6.2) provide a two level means for naming aggregates of objects. The first level is the aggregate name (set name) specified using normal NSW object naming conventions. The second level names a member of the aggregate, and may be specified using NSW naming conventions or host specific naming conventions, depending on the type of set file. A distinguishable special symbol "!" is used to separate the set name from the object name within the set. In the case of native host set files, "!" serves to mark the transition from NSW conventions to host conventions. Files within a workspace can be named in this fashion when interacting with NSW command interpreter software. The set name denotes the workspace and the object name devotes the workspace file.

Names for NSW nodes are similar in format to object names with the convention that person node names have name components corresponding to the person's last, first, and middle names, followed by an integer to ensure uniqueness. Organization node names include the convention that the ordered name components specify the node creation ancestry (Section 10).

Instantiating services in workspaces adds a further conceptual level to naming. Services are run with their name context automatically set to the service workspace. Much like the basic NSW naming conventions, workspace conventions have an "unscope" mechanism that allows users to specify objects outside of the workspace domain. The special symbol "^" is used to indicate that the named object is to be interpreted in the context of the user's NSW session rather than in the workspace context. This particular "unscoping" mechanism is available only when interacting with NSW mode services or with a WS-CI.

NSW supports mechanisms which enable users to disambiguate user supplied object specs which refer to more than a single catalogue object. Users can control the operation of the lookup function in two ways. There are NSW commands for altering the scopes in effect, and attributes which the specified object must possess to be considered for a match can be supplied as part of the object spec. These means for controlling the lookup function are available only when operating within the domain of the NSW name space. Considerably less sophisticated means are provided within workspaces, and WS-CIs do not support such complex partial name to complete object name mechanisms.

15. System Software Architecture

This section discusses the structure of the software that implements the NSW design by considering three important aspects of the system software.

1. Major functional modules.
2. User access points.
3. Intercomponent protocols.

A detailed discussion of the software is beyond the scope of this document.

15.1 Major Functional Modules

The NSW 4.1 implementation defines a functional modularization of the NSW software. Major changes to that functional modularization will not be required to implement the design documented here, although the capabilities of some modules will need to be enhanced substantially. This section first reviews the major system modules, and then suggests how they might evolve to support the new design.

NSW Core Software

Works Manager (WM) - The works manager manages the NSW resource catalogue, node permission data base and session record data base. It is the major part of the core software. The WM runs on a single host, and other system components request its services by sending it messages. While one might well question the wisdom of centralizing the core functions and supporting data bases in this way, it would be a major undertaking to distribute the WM's functions and the technology for doing so is not well understood. We, therefore, make no such recommendation.

Interactive Batch Submission (IBS) program - When a user invokes a batch service this module interacts with him to gather the information required to submit the job (e.g., input files, output file names, etc.). After gathering the information IBS constructs the necessary JCL and places the job in an NSW job queue. The job is transmitted from this NSW job queue to the appropriate batch processing host.

Works Manager Operator (WMO) - WMO manages the NSW batch job

queue. It submits jobs from the queue to batch processing hosts and notifies users when their jobs have been processed and the results are ready.

NSW Host Supporting Software

Foreman (FM) - The foreman is responsible for managing NSW services on its local host. It starts a service for a user at the request of the WM, controls the service as it executes interacting on its behalf as necessary to provide it an NSW execution environment, and terminates the service when the user is finished with it.

File Package (FP) - The file package is responsible for managing NSW files stored on its host and for moving files into and out of its host. To move a file from one host to another, the file packages on the two hosts cooperate, translating the file data and structure as necessary, to ensure that it can be stored and used on the destination host.

User Access Software

Front End (FE) - The front end is the user's interface to the NSW. It acts as an NSW command interpreter (NSW-CI), accepting user commands and interacting with other NSW components to accomplish the commands. In addition, it manages its user's communication with interactive services.

The software modules identified above are realized as processes. Each active user is represented by a FE process, and each interactive tool is represented by a FM process. The operation of the WM and FPs are more transaction oriented, and WM and FP processes are allocated as needed to satisfy requests made by users and executing services.

The communication between processes is provided by an interprocess communication facility named MSG. MSG supports three modes of communication: process-to-process messages, direct process-to-process communication paths (e.g., ARPANET host-host protocol connections, ARPANET TELNET connections), and process-to-process alarms (short, high priority signals which bypass the normal message flow control and buffering disciplines). In addition, MSG supports two types of process addresses. A generic address specifies any process from a process class; e.g., a WM process, a FM process at a particular host. A generic address is typically used when the sending and receiving processes have not previously communicated as, for example, to initiate a transaction. When a user attempts to

login, the FE process acting on his behalf sends a generically addressed message to a WM process which contains the user's login name and password. The second type of address is a specific address which identifies a particular process; e.g., FE process #27 at a particular host. It can be used when the sending process and receiving process have engaged in previous communication. For example, the WM process allocated to handle a login transaction for a user will send a specifically address message to the user's FE process to inform it of the outcome of the login attempt.

Figure 15-1 illustrates the principal software modules and their relation to one another. Interactions between the software modules are governed by a set of NSW intercomponent protocols. For example, there is an intercomponent protocol for starting a service.

For the current NSW implementation the NSW core software runs under the TENEX and TOPS-20 operating systems. NSW host supporting software has been implemented for the Multics, OS/360, TENEX and TOPS-20 operating systems, and is under development for the Univac 1100 computer system. User access software in the form of NSW FEs has been implemented for the TENEX and TOPS-20 systems, and is under development for the Unix operating system.

We next suggest five ways in which this software architecture should change to support the design described in this document.

1. Addition of the Workspace Command Interpreter (WS-CI) function.

Sections 8 and 9 discussed the WS-CI notion. In our view, the WS-CI function should be part of the NSW host supporting software, and hosts should have the option of supporting it or not. Although the details of the relation of a WS-CI to the foreman component may be expected to vary from host to host, the WS-CI is conceptually part of the foreman. Another way of saying this is that there shall be no WS-CI generic process class. When a WS-CI on a particular host is invoked as a service, a FM process will be started on the host to manage the WS-CI, and all WS-CI requests for NSW operations will be initiated by the FM on behalf of the WS-CI. Thus, the principal impact of the WS-CI notion is on the foreman component. Other NSW components interacting with a FM need not be concerned with whether the FM is acting on behalf of a program service or a WS-CI.

2. Addition of file package capabilities to the foreman.

This change is motivated by performance considerations.

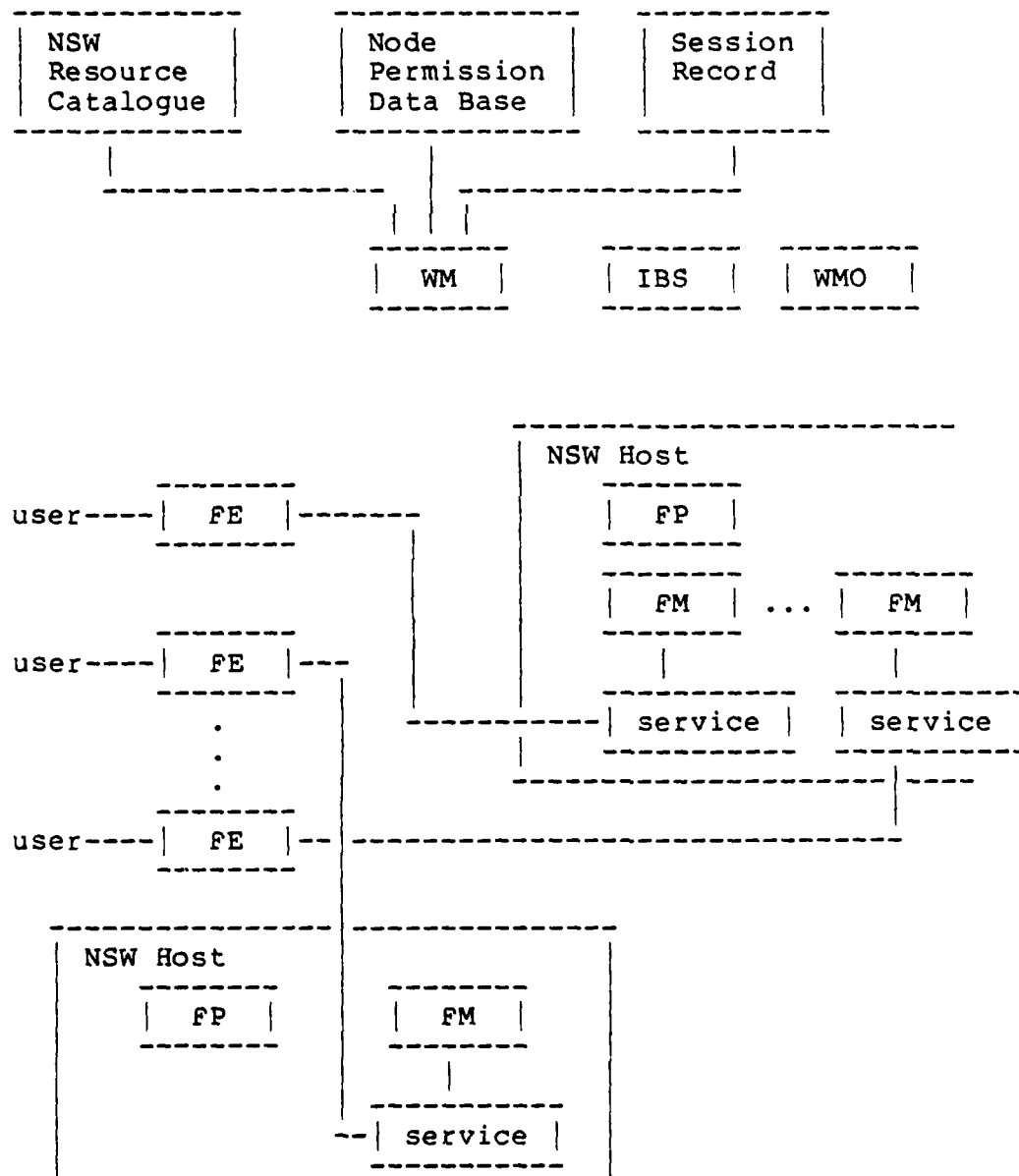


Figure 15-1:

In the current implementation all movement of files between NSW file space and workspaces requires participation of a FP process on the workspace host.

For example, when a program service attempts to access an NSW file, its FM sends a message to the WM requesting access to the file. The WM sends a message to the FP on the service host instructing it to place a copy of the file in the service workspace. After placing a copy of the file in the workspace (this involves interacting with a FP on another host if the file is remotely stored), the FP replies to the WM by means of a message, and the WM informs the FM that the file is ready to be accessed by means of a message. If the FM able will to act as a FP, two messages (the FP->WM and WM->FM) and the associated delays could be eliminated, resulting in improved responsiveness for the file movement operation. Similar improvements could be realized in moving files from workspaces to NSW file space. Implementing this change involves three things:

- o Adding FP capability to the FM. We suggest that several levels of FM file package capability be defined including
 - . no FP capability in the FM;
 - . ability to perform FP operations that are local to the FM host;
 - . ability to interact with non-local FPs to accomplish file movement.

and that hosts be permitted to choose which to support:

- o Modifying the FM/WM protocol for file access and delivery operations to permit the FM to specify the extent, if any, to which it is willing to act as a FP.
 - o Modifying FPs to be willing to engage in interhost file movement with processes which are not FPs.
3. Addition of a (limited) file package capability to the front ends.

Several desirable user features, including command procedures (the ability to specify "macro" commands which can be stored as NSW files for subsequent invocation from the NSW-CI), the "type" command (the ability to cause a text file to be typed on the user's terminal), and the typescript feature (the ability to record the user's terminal session - type in and type out - in an NSW file for later analysis), require a

capability to move files between NSW file space and user FEs.

The FE should be able to initiate file movement in the same way a FM does. Similarly, it should be able to act as a FP as a FM can. That is, the changes recommended above for FM/WM/FP interactions should also apply to FE/WM/FP interactions.

4. Addition of FTP capability to the NSW core software.

Adequate support for ICP services (see Section 8) will require that users be able to move files between NSW file space and file systems of hosts that support ICP services, many of which may not support NSW file packages. This file movement will be supported by the augmentations to the import/export functions described in Section 12.3. As noted there, these require that the core software support standard ARPANET file transfer protocol.

There are several ways that an FTP capability might be added to the core software. First, we observe that, as currently implemented, the import and export functions collect the information required to interact with a FTP server on a non-NSW file host. This means that no modifications are required to the user interface to the import and export functions implemented by the FE or to the FE/WM part of the protocols that accomplish the import and export functions. One way to incorporate FTP would be to modify the file package on the works manager host so that it can use FTP when necessary to move files between NSW and non-NSW file hosts. To import a file from a non-NSW file host the WM would instruct that FP to obtain the file and store it on the WM host. The FP would use the FTP protocol to interact with a FTP server process at the file site rather than using the NSW FP protocol to interact with another FP. To export a file to a non-NSW file host, the file would first be moved to the WM host (if the file were already there, no action would be required; otherwise existing mechanisms could be used to move it) and then transmitted by the FP on the WM host to the specified destination. The extra file transfer required to get the NSW file to the host with the FTP file package could be eliminated if other FPs supported FTP.

5. Modification to the user access point.

These modifications are the subject of Section 15.2 below

15.2 User Access Points

As described above, the NSW 4.1 implementation supports user access to NSW by means of front end processes. To access NSW a user first obtains a FE and then logs in.

FEs may be accessed by users only by means of an ARPANET ICP exchange. In this sense, obtaining a FE 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 FE 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 a FE. A front end obtained in this way is known as a "dispatched" FE.

After obtaining a FE process, the user interacts with it through the ARPANET TELNET connection established by the ICP exchange. The FE 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 FE and the other between the FE and the service.

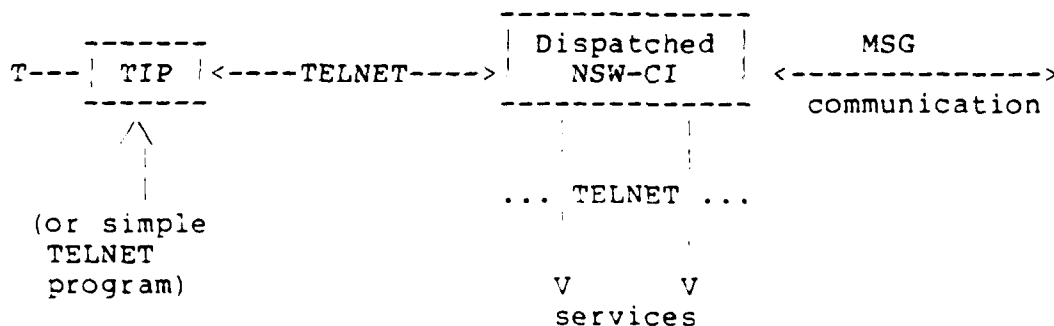


Figure 15-2:

This configuration is shown schematically in Figure 15-2. 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.

A more optimal approach would be to place the FE on the user's access host. There are several difficulties with adopting this approach for all user access.

1. The FE interacts with other components by means of MSG. Each access host would need an implementation of MSG, which is a substantial undertaking, and which would be out of the question 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 ensure uniform implementations.

This section considers some alternate approaches to user access. The intent is that the system evolve to support these alternatives in addition to the current approach.

The first alternative is the one introduced above: having the front end run in the user's access host. This 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. The Unix NSW front end is an example of this alternative.

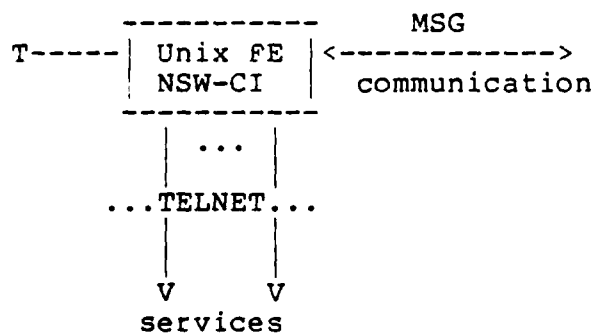


Figure 15-3:

The configuration for this alternative is shown in Figure 15-3.

The Unix FE behaves much as a dispatched NSW-CI in terms of its relation to other NSW components and to services. However, the fact that it is not dispatched but may be run directly by a user

32

raises the issue of the "security kernel" of the NSW system. Until the introduction of the Unix FE, a user could not tamper with any of the system components. Thus, the security kernel included all components, and the FE could be regarded as a trusted component. Since it will be impossible to guarantee that a user has not tampered with a Unix FE process, this is no longer the case. As far as we can tell, for the current NSW system the fact that the Unix FE cannot be regarded as a trusted component has no impact. However, if the FE is enhanced with file package capability along the lines suggested above, care needs to be taken in the detailed design for implementing the capability to ensure that file access controls are not subverted. For example, file movement is currently initiated by the recipient file package by means of a "sendme" request to the sending file package that specifies the file to be moved. The donor file package assumes the recipient is properly authorized to access the file; this will not necessarily be the case if the recipient is a non-dispatched FE process (since there is no guarantee that a user has not tampered with the FE to cause it to access files without WM authorization).

33

The second alternative approach is based on the observation that the FE performs two somewhat independent functions: parsing and interpreting user commands (parser function); and, managing user o-service communication (switcher function). This suggests that the FE could be implemented by a parser module (NSW-CI) and a switcher module. The switcher module could reside on the user's access host, the parser on an NSW core host, and communication between them could be supported by TELNET communication.

This configuration is illustrated in Figure 15-4, and would work roughly as follows. The switcher or TAP (for Terminal Access Process) would be a sophisticated TELNET program capable

32

We use the term "security kernel" informally here to mean the part of the system which may be trusted to adhere to system access control and security conventions. An operational definition that is adequate for our purposes is that the security kernel of NSW contains all those components a user cannot tamper with.

33

Originally suggested some time ago by Kirk Sattley and Steve Warshall.

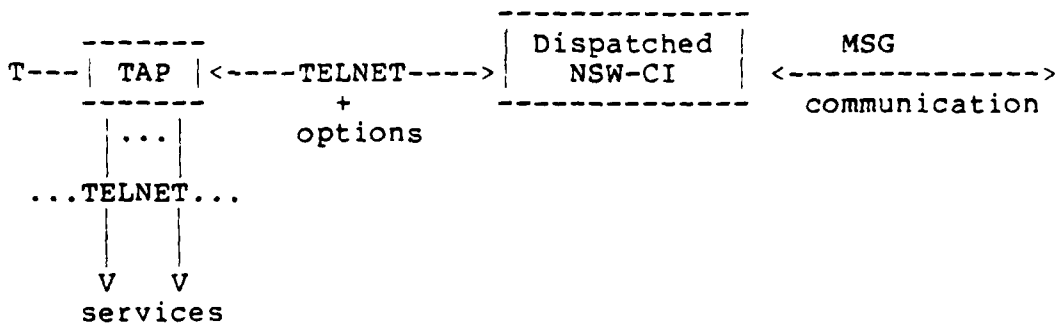


Figure 15-4:

of maintaining multiple connections and able to interact with the NSW-CI by means of a TAP/NSW-CI protocol. The TAP/NSW-CI protocol would be implemented by means of TELNET options, and would be used to coordinate the establishment and breaking of user-to-service connections.

To access the system the user would start up the TAP which would engage in an ICP exchange with a NSW-CI host to obtain a dispatched NSW-CI. To alert the NSW-CI that it is communicating with a TAP rather than a simple TELNET, the TAP initiates a TELNET option exchange ("WILL TAP" - which is part of the TAP/NSW-CI protocol) with the NSW-CI. The TAP would pass NSW commands to the NSW-CI using the TELNET connection. When a service is invoked the user-to-service path would be established between the TAP and the service rather than between the NSW-CI and the service. To do this the protocol for starting a service must be modified so that the service's FM can be told the desired termination for the user-to-service communication path (i.e., NSW-CI or TAP). In addition, the NSW-CI would use the TAP/NSW-CI protocol to instruct the TAP to establish the communication path to the service.

Since the NSW-CI is dispatched, it can be considered part of the security kernel. The Terminal Access Process can not, in general, be considered part of the kernel.

For users who access the NSW through a host that implements a Workspace Command Interpreter (WS-CI), a viable approach is for the WS-CI to also act as the users NSW-CI. This configuration is illustrated in Figure 15-5. Architecturally, it is similar to the Unix FE in that the WS-CI would interact with the rest of the system by means of MSG, and that it could not be considered part of the security kernel. The details of this approach, such as

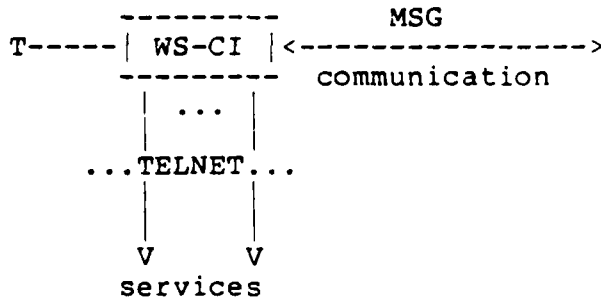


Figure 15-5:

the relation (if any) between a WS-CI used in this way and an associated foreman, need to be worked out.

At this point we take a slight digression and point out an additional similarity between the Unix FE and WS-CIs. Unix is a powerful host operating system which supports a variety of sophisticated services, and it is reasonable to allow (properly authorized) users whose access point to NSW is a Unix FE to access these Unix services through their Unix FE. Such services would run under the control of the Unix FE much as other NSW program services run under the control of a foreman/WS-CI. Further design work is required to specify the details of how services in Unix FE hosts (or other hosts that provide NSW FEs) will be made available through NSW. For example, supporting services on Unix FE hosts will require means for moving files between Unix and the NSW. We have already discussed means for accomplishing this sort of file movement in Sections 12.3 and 15.1. In addition, means for starting, stopping and otherwise controlling such services need to be designed.

Finally, it is possible to imagine a transaction oriented mode of user access. This is illustrated schematically in Figure 15-6. The idea is that any process that observes the protocols should be able to initiate any NSW operation. Of course the process would not be part of the security kernel, and so the message from the process that initiates the transaction would carry the user node and password along with information about the operation requested. This would enable the core system to authenticate the process in order to establish its authorization to perform the requested transaction.

MSG

process <-----> NSW system components
communication

Figure 15-6:

15.3 Intercomponent Protocols

As noted in Sections 3 and 15.1, there are a set of intercomponent protocols which specify the interactions between system components that implement the various functions provided by the system.

There are actually two distinct protocol layers here. There is an NSW Transaction Protocol (NSWTP) which defines the general rules for intercomponent interactions. NSWTP is concerned with issues common to all NSW transactions, such as the format of interprocess messages, the conventions for reply messages, and so forth. The second protocol layer defines the particular messages and message exchange patterns that implement each system function, and has come to be known as the "NSW protocol scenarios". Its concern is with issues such as the information that must be exchanged between components to accomplish various functions, the components that must participate in the functions, and the sequences of interactions that are required.

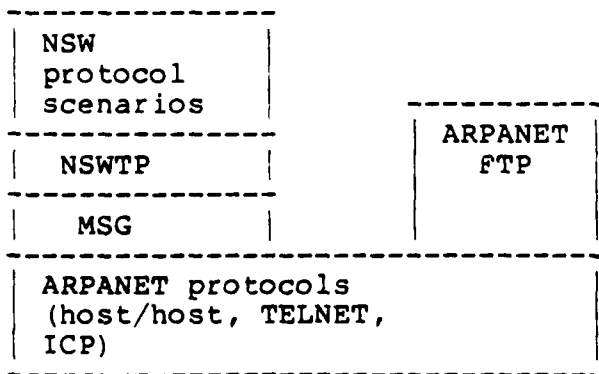


Figure 15-7:

The relation of these protocol layers to one another, and to MSG and the ARPANET protocols is shown in Figure 15-7.

Implementation of the design presented in this document will require some modifications to NSWTP and to the NSW protocol scenarios. This section discusses changes to NSWTP. Changes to the NSW protocol scenarios, such as those required to permit file package capability to be added to the foreman or division of the front end into parser and switcher components, will be described in other documents.

The following are suggested changes to NSWTP.

1. Transaction ID.

Each protocol message should carry an identifier that identifies the transaction of which the message is a part. The transaction ID would be assigned by the process that initiates the protocol transaction, according to system-wide rules for transaction ID generation. At present NSWTP does not quite support transaction IDs of this sort. For 2-party request/response type transactions there are transaction IDs, but for more complex transactions, as for example those involving multiple parties, there are not. Rather, each 2-party interaction that is part of such a transaction has its own ID. The absence of a transaction-wide ID restricts the message exchange patterns that NSW protocol patterns can use.

It may be desirable to consider an enhancement to MSG to support a third class of message addressing (in addition to generic and specific) called transaction oriented addressing. A transaction oriented address would be an address of the form:

The process at host X in the generic class Y
that is responsible for transaction Z.

This type of address would be useful in situations where it is necessary to communicate with a process handling a transaction but where its exact name is unknown. For example, it could be used for messages requesting status information about a transaction when the message initiating the transaction had not been acknowledged.

2. Reply control.

Most of the NSW protocol scenarios involve one or more reply messages. NSWTP should include a reply control convention which permits the "request" messages to

specify where reply messages are to be sent. It should be possible to specify the following:

- a. no reply is required
- b. reply to sender
- c. reply to specified process.

3. Determining the system security kernel.

A process that receives a message requesting a certain action needs to be able to determine whether the message sender is to be trusted. Typically, it must make this determination in order to decide whether or not to expend the system resources required to perform the action or to give the requesting process access to information it has requested. For example, when file package capabilities are added to front end processes, a file package receiving a "sendme" request must decide whether the requestor is part of the security kernel.

We suggest that MSG be modified as follows to facilitate this decision making. MSG currently delivers the address of a sending (requesting) process along with the message (request) data. The modification would be for MSG to also deliver whether or not the sending process is part of the security kernel. This would be implemented by a minor change in the process-to-MSG interface. MSG would decide whether or not a particular process was in the security kernel based on declarations made in the configuration data base it uses and which is maintained by systems operations personnel. System personnel would declare certain (generic class, host) pairs as trusted. For example, dispatched front end processes on TOPS-20 hosts might be declared as part of the security kernel but not front end processes on Unix hosts.

4. Authorization for non-kernel processes.

Because not all components engaging in a transaction may be part of the trusted security kernel, some components may need to be authenticated by others. For example, before a file package responds to a "sendme" request from an (untrusted) PE process, it should have authorization to do so from the WM.

This authentication and authorization can be accomplished by means of a message to the "service providing" component from an authenticating process that specifies the name of the process from which to expect and to accept a specified request message. Before acting on the request, the recipient can compare

the sender's address with that supplied by the authenticating process. The authenticating process would, of course, be within the security kernel; for example, it might be a WM authorizing a FP to accept a "sendme" request from a specified FE process.

This procedure is similar to that currently used by the FE to ensure it has established contact with the correct FM when an interactive program service is started. The details of how this procedure would be integrated into the various protocol scenarios will be described in other documents.

5. Help Calls

The completion of some transactions may require information in addition to that specified when the transaction was initiated. Since the additional information may not be needed in every transaction of the requested type it may not be supplied when the transaction is initiated. Furthermore, the need for the information may not be apparent when the transaction is initiated. That is, it may only be detected later by a process several messages removed from the initiating process.

In order to make it convenient for the process needing the information to obtain it, NSWTP supports the notion of a "help call". Each message in a transaction that initiates an action that may require additional information should specify the name of the process (a "help process") which may be called for any additional information. When a process detects that it needs additional information it can obtain it by sending a "help" message to the appropriate help process address. The help process is expected to respond to the help message by supplying the requested information.

While the NSW protocols currently support help calls, they do it in a somewhat restricted fashion. In particular, a help process name is not specified. Instead, a small numeric value is used as a code to specify which of a small number of processes may be called for help.

6. Authentication Information.

In order to allow transactions to be initiated by processes that are not logged into NSW, NSWTP should permit the message that initiates a transaction to carry with it authentication information. The authentication information would be supplied as an

optional field of the NSWTP message. It would be specified as an optional field of the NSWTP message and would include the user node specification and the password for the node.

7. Multiple functions

As an optimization, NSWTP should permit multiple functions to be invoked by a single MSG message.

16. Implementation of NSW Host Supporting Software

The design presented in this document provides for a wide range

34

in the extent to which hosts may participate in NSW. The intent is to give host personnel a choice in the level of their host's commitment to NSW. This is a significant departure from NSW 4.1 where there is little room for such choice, and the decision to participate in NSW required a large commitment in terms of software development. In particular, to be a service bearing host in NSW 4.1, it is necessary to implement MSG, a foreman and a file package.

This variation in the level of host participation derives from the variability of services and support encompassed by the new design: native and NSW services, ICP services, WS-CIs, permanent workspaces, and so forth.

This section identifies some meaningful choice points for host participation. Other levels of participation are possible also. A more detailed analysis of host supporting software is required to precisely specify a "standard" set of levels.

A major characteristic of an "NSW" host is whether it implements MSG or not. If not, it cannot provide any other NSW supporting software since the components interact by means of MSG. However, such a host can support ICP services and devices that are accessible from NSW. The services and devices that are to be accessible must be entered into NSW resource space by a process of registration (see Sections 7 and 8). Furthermore, to support the use of the devices from NSW the system core software must implement the device access procedures (see Section 7). Files referenced and created by ICP services can be moved between NSW file space and ICP services by means of the import and export functions.

For hosts that implement MSG there are a number of choices possible regarding the NSW support to be provided. These include the following.

- o A host could choose to implement MSG and a file package. Such a host could serve as a file storage host.
- o A host could choose to implement MSG and a foreman. A host implementing only a foreman could support program

34

The focus in this section is host supporting software (See Sections 3 and 15.1.)

services. Files referenced and created by the services could be moved between the NSW file system and the service workspace (imported and exported) by means of FTP (see Sections 12.3 and 15.1). There is a choice with respect to the foreman implementation regarding whether the foreman will support native services, NSW services or both. In addition, a host implementing a foreman can choose to support a WS-CI or not.

- o A host could choose to implement MSG, a file package, and a foreman. In this case, all the choices with respect to foreman implementation noted above are possible, and in addition, the host can choose whether to support permanent workspaces or not.

The discussion above has not considered batch services. To provide NSW batch services a host must implement a modest Batch Job Package, MSG and a file package. We believe it would be useful to investigate the possibility of allowing hosts to support batch services without requiring MSG and file package implementations on the hosts.

I. APPENDIX: On the Implementation of Workspaces from Host User Slots

NSW supporting software on the hosts provide the following functions which NSW core software can invoke.

1. Host-Register-US

35

Host-Register-US (usage-spec, user-slot-inf)
-> husid [,hwid, ws-data]

This function is typically called when a user attempts to register an external user slot for use as a permanent NSW workspace, but may be called in other situations, such as when NSW operators add a set of new user slots to the pool of slots for a host. It serves to make the host user slot specified by "user-slot-inf" known to the NSW system and available for subsequent use as specified by "usage-spec". It does this by making additions to user slot data bases maintained by NSW core software and by NSW host supporting software (see below). In the case of registration of an external user slot for use as a permanent workspace, this function would be followed by calls on Make-WS (to create a workspace descriptor for the user slot from hwid and ws-data; see below) and Enter.

There is a matching Host-UnRegister-US (husid) operation.

"User-slot-inf" is host dependent information supplied to identify the user slot on the host and to establish the caller's right to access it. For example, it might include a directory name, a password, etc.

"Usage-spec" may be:

- a. general, meaning the host may use the user slot as it sees fit to meet NSW workspace and file storage requirements.
- b. for workspace use only.
- c. for file storage use only.
- d. for permanent workspaces only. In this case,

35

Items in square brackets, "[" and "]", are optional

there is an additional parameter = do/don't allocate now. If allocation is specified, the operation returns a host specific id (hwid) for the new workspace.

The values returned by this operation are a host user slot id (husid), and (optionally) a host workspace id (hwid) and workspace data (ws-data).

A host user slot id (husid) is an id whose format is uniform across all NSW hosts. It serves to identify a particular user slot on a host. NSW supporting software on hosts is expected to maintain a correspondence between husids and user slots.

A host workspace id (hwid) and workspace data (ws-data) are returned only if the usage-spec specifies use as a permanent workspace. A hwid is an identifier that can be used to specify a particular workspace to NSW software on the host (for example, to specify a workspace when a service is instantiated). Its format is uniform across all NSW hosts. It is used by NSW much as a host specific file name is used. Just as a host specific file name identifies a file on a host, a hwid identifies a workspace on the host. A difference is that files are objects provided by the host operating systems whereas workspaces are not, and must be implemented, to a large degree, by NSW supporting software on the host.

The workspace data (ws-data) returned includes information about the workspace, in addition to its host and hwid, from which a workspace descriptor can be created.

It is useful to think of the semantics of Host-Register-US in terms of its effects on two user slot data bases, one maintained by NSW core software and the other by NSW supporting software on the host of the user slot.

The core NSW table would be indexed by a system-wide user slot id (usid), and would contain the information shown in Table 16-1. The host table would contain similar information and would be indexed by husid (see Table 16-2).

2. Host-Allocate-WS

Host-Allocate-WS () -> hwid, ws-data

This function is used by NSW core software to allocate

```

usid ----> host
            hsuid
            allowed usage (one of):
                general
                file storage only
                workspace only
                permanent workspace only
                (password req'd to access
                 internal/external user slot ...)
            current usage

```

Table 16-1:

```

husid ---> corresponding host user slot
            allowed usage (one of):
                general
                file storage only
                workspace only
                permanent workspace only
                (password req'd to access
                 internal/external user slot ...)
            current usage

```

Table 16-2:

a workspace on the host. There is a corresponding Host-DeAllocate-WS (hwid) operation.

3. Host-Allocate-Perm-WS

Host-Allocate-Perm-WS () -> husid, hwid, ws-data

This function is used to allocate a permanent workspace on the host. The return parameters identify the user slot for the workspace (husid), the workspace allocated (hwid), and workspace data (ws-data). There is a corresponding Host-DeAllocate-Perm-WS (hwid, husid) operation.

4. Host-Instantiate-Program-Service

Host-Instantiate-Program-Service (WS-spec,
host-service-id, ...)

-> hwid, ws-data, communication-path-spec, ...

This function is used to instantiate a service on the host. WS-spec is a workspace specification. It may be:

- a. an hwid, in which case the NSW core software specifies a particular workspace to be used.
- b. don't care/temp, in which case the NSW core software wants one allocated from the pool on the host. When the service is normally terminated, the workspace should be returned to the pool.
- c. don't care/long term, in which case the workspace should be allocated from the pool, but not returned to the pool until an explicit call is made by the NSW core software to do so.

Host-service-id specifies the program service desired. The communication-path-spec return value specifies the communication path to be established between user and service for supporting user/service interactions (see Section 15.2).

The NSW workspace operations implemented by the NSW core software are built out of the NSW support software operations just described. The NSW core software implements the following workspace operations:

1. Make-WS

Make-WS (host, hwid, workspace-data)
-> workspace-descriptor

This operation creates a workspace descriptor from the parameters supplied. When (if) the workspace is entered into NSW name space, the workspace descriptor created by Make-WS is the object descriptor stored in the i-node.

2. Allocate-WS

Allocate-WS (host) -> workspace-descriptor

This allocates a workspace on the specified host by calling the Host-Allocate-WS () routine implemented by NSW supporting software and using the Make-WS routine to create the workspace descriptor it returns. As above, the workspace descriptor is an object descriptor for the allocated workspace that can be stored in an i-node when the workspace is entered into NSW name

space. The workspace descriptor includes the host of the workspace and the hwid returned by the Host-Allocate-WS () routine, in addition to other information about the workspace.

3. Allocate-Perm-WS

Allocate-Perm-WS (host) -> workspace-descriptor

This allocates a permanent workspace on the host, and returns a workspace descriptor for the workspace.

4. Instantiate-Program-Service, Instantiate-WS-CI

Instantiate-Program-Service (service-descriptor
[, workspace-descriptor])
-> status, workspace-descriptor,
communication-path

Instantiate-WS-CI (...) -> ...

These routines are used to start program services and WS-CI services, respectively. The workspace descriptor argument may be omitted, in which case the system allocates a temporary workspace for the service. The Instantiate-Program-Service function makes use of the Host-Instantiate-Program-Service function implemented by NSW supporting software to start the service on the service bearing host.

NSW VERSION 6.0 ENHANCEMENTS SPECIFICATION

Prepared by
Bolt Beranek and Newman Inc.

D R A F T

February 1981

Appendix C

This appendix is a preliminary and in some areas incomplete draft version of the design and specification of those system enhancements scheduled for NSW release 6.0.

The enhancement tasks include the following:

- o Revised and Integrated NSW Resource Catalog Design and Implementation
- o Decentralized Protocols for NSW File Movement and Service Activation
- o Coordinated System Timeout Control
- o Automatic Database Initialization on Re-login
- o Limited User I/O Commands
- o Extended Services and Service Session Support
- o User Oriented Status Commands
- o TBH/WM Data Base Synchronization Improvements
- o Programmable Control Signals for FE-tool Communication

TABLE OF CONTENTS

	Page
1 GENERAL FORM OF OBJECT NAME	1
2 KEYS	1
3 OBJECT ATTRIBUTES	2
4 NAME LOOKUP	2
4.1 FULL PATH LOOKUP	3
4.2 ELLIPSES	3
4.3 SCOPING	4
5 ENTERING CATALOG OBJECT NAMES	7
6 NAMING GROUPS OF OBJECTS	9
7 ASPECTS OF THE FILE SYSTEM MODEL	11
7.1 FILE IMAGES	11
7.2 IMMOVABLE OBJECTS	12
7.3 DIRECT FILE ACCESS	13
7.4 FILE SEMAPHORES [locks]	13
8 NEW OBJECT TYPES	16
9 OBJECT TYPE-SPECIFIC ASPECTS OF THE RESOURCE CATALOG	18
10 FILE ACCESS and IMAGE MAINTENANCE PROTOCOL ENHANCEMENTS	21
10.1 Enter-File-Object	21
10.2 Access-to-File-Object	23
10.3 File-Image-Update	28
11 AVOIDING TIMEOUT ON LONG FILE TRANSFER	30
11.1 Intermediate Status Replies	30
11.2 Handling Timeouts	32
11.3 Intermediate Status Reply Specification	33
11.4 Status List Specification	37
12 DISTRIBUTED "ARE YOU THERE"	39
12.1 Status Probes	39
12.2 Status Probe Specification	40
12.3 User Interface for Status Probing	45
13 Specification of User Interface Changes	46
13.1 Conventions	47
13.2 SHOW Command	51
13.2.1 SHOW Commands which Initiate Remote Query Requests	51
13.2.2 SHOW Commands which Do Not Initiate Remote Query Requests	61
13.3 SET Command	62
13.3.1 SET Commands which Generate Remote Activity	63
13.3.2 SET Commands which Do Not Generate Remote Activity	65
13.4 MODIFY Command	66
13.5 Other New Commands	68

	13.5.1	DELETE, RENAME, and COPY Commands	68
	13.5.2	PLACE Command	68
	13.5.3	TYPE, PRINT, and GET Commands	69
	13.5.4	USE Command	69
	13.5.5	SAVE Command	71
	13.5.6	ABORT and TERMINATE Commands	71
	13.5.7	LOGIN Command Interface Change	71
	13.6	Command Tree of Changed Commands	72
14		Specification of Associated NSWTP Changes	73
	14.1	Conventions	74
	14.2	Messages which Span Physical Message Boundaries	75
	14.3	WM-SHOW-NAMES	76
	14.4	WM-SHOW-DESCRIPTOR	76
	14.5	WM-SHOW-PUBLIC	78
	14.6	WM-SHOW-NODE	79
	14.7	WM-SHOW-SESSION	79
	14.8	WM-MODIFY-DESCRIPTOR	80
	14.9	WM-MODIFY-NODE	81
	14.10	WM-MODIFY-SESSION	81
	14.11	WM-DELETE	82
	14.12	WM-COPY	82
	14.13	WM-RENAME	83
	14.14	WM-PLACE-IMAGE	83
	14.15	Starting Up a NSW Service	84
	14.16	WM-LOOKUP-FILE	85
15		TBH/WM DATA BASE SYNCHRONIZATION	86
16		AUTOMATIC CLEANUP OF EXISTING LOGIN SESSION RECORD ON RE-LOGIN ATTEMPT	88
17		WORKSPACE COMMAND INTERPRETERS and GENERALIZED SERVICE SUPPORT	90
	17.1	User Interface Changes for Service Control	92
	17.2	CHAINING TOOLS IN EXISTING WORKSPACE	92
	17.3	DETACH SERVICE SESSION	94
	17.3.1	FE-CLOSECONN	94
18		FILE-HANDLING CAPABILITIES for the UNIX FE	95
	18.1	User Interface	96
	18.1.1	TYPE command	96
	18.1.2	PRINT command	97
	18.1.3	GET command	97
	18.2	Importing the File to UNIX	98
	18.2.1	The Protocol Scenario	98
	18.2.2	Implementation	99
	18.3	Viewing/Listing the File	100
	18.3.1	Design Aspect	100
	18.3.2	Implementation	100
19		MISCELLANEOUS OTHER CHANGES	101

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 useful to view an object's name as the name for a hierarchical path through NSW name space to the object. A complete pathname refers to a name beginning at the implied root of the NSW hierarchical name space and uniquely naming a single terminal object through the sequence of ordered name components.

2 KEYS

NSW access control is based on permissions (capabilities) held by the accessing agent. For controlling access to resource catalog objects, a permission (a key) may refer to either a single unique object in NSW name space, or all objects in an entire region of NSW name space.

Syntactically we write:

ABC.DEF.G as indicating a permission referring to the object uniquely designated by ordered name components ABC, DEF and G. The object need not exist at the time the permission is created.

Or ABC.DEF* as indicating a permission referring to the entire region of name space containing objects whose complete pathname begins with ABC.DEF. The region is evaluated at each access and includes all objects in the region at the time the right is exercised.

Regions of NSW name space designating a permission may include only a single instance of the "*" notation, anywhere in the name (to support simplified implementation). For example, A.B, A.B*, and A*B would all be legal keys, but A*B* would not. The special key "*" refers to the entire NSW name space.

There are different keys for governing different kinds of access privilege. Three kinds of keys are defined for reading and writing the catalog itself. These keys are:

- o LOOKUP keys, required to do catalog object lookup operations in a given region (somewhat equivalent to directory read access on some systems),

- o ENTER keys, required to create a new object name in a given region (i.e. directory write),
- o DELETE keys, required to remove a current catalog object name in a given region (a specialized form of directory write).

These keys are applicable to all catalog operations regardless of the type of the object being manipulated.

Keys are stored with the node record to which the permission applies. 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 rights are determined 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". For now, the public key table is modified only via system administrative means).

3 OBJECT ATTRIBUTES

All objects in the NSW resource catalog have attributes including (but not limited to) "type" and "site", where "type" is currently envisioned to be one of file, device, workspace, or service, and "site" indicates the host on which the object resides (site may be a list of locations in special cases). Every NSW object must have a unique name independent of its attributes. Objects cannot differ only in their attributes. Syntactically we can write:

A.B.C.D/type=device to refer to an object with name A.B.C.D. which is of type device or

A.B.C.D.E/type=file;site=ISIE to refer to an object of type file and which is stored on host ISIE.

4 NAME LOOKUP

In general, manipulation of NSW resource catalog objects proceeds in three phases:

- o name and attribute lookup
- o general disambiguation (optional)
- o typed access control check

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. This set can be reduced to a single object via a search rule or user help. 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.

4.1 FULL PATH LOOKUP

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 full pathname is specified:

- o Verify 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.
- o 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 full path name for the object. 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. Object names employing either or both of these mechanisms are referred to as object specifiers (or object specs, or just specs).

4.2 ELLIPSES

When specifying an object name, the special symbol "... " may be used to indicate 0 or more components may be missing from the name as specified. Missing components may be before the

first specified component, between specified components, or after the last specified component, with as many instances as required.

For example:

A.B...E, meaning 0 or more missing component names between the B and E component names.

A.B..., meaning possible missing component names after the B component, etc.

...A.B.C

...A.B...C.D...

A...B...C.D...E

are all legal object specifiers.

When an object specifier includes ellipses, the areas of NSW object space to which the user has lookup keys are searched for possible matching entries. The set of matching name entries is then reduced by any attribute requirements associated with the lookup; in particular, if the lookup specified a certain object type, only names with that object type are considered a match. If the set of matching objects has 0 members, the lookup fails. If the set of matching objects has exactly one member, that member is used in completing the operation (i.e. the operation specific access control check and subsequent object manipulation). If the object matching set has at least two members, user or program help can be invoked (if requested) to select a single member, which is then processed as above. If no help is provided when there are multiple matches, the lookup fails as it is an ambiguous object reference.

We refer to a name which includes ellipses as an incomplete name specifier, whereas a name which does not contain ellipses is a complete name specifier.

4.3 SCOPING

The resource catalog lookup function allows users (programs) to reference objects relative to user selectable regions of NSW resource space. These regions which are used as part of the lookup function are called scopes. A scope is a sequence of name components that designates a region of NSW name space in which user supplied specs are to be looked up or entered. A scope is a full path region specifier with exactly one component of variability occurring at the end. For example, A.B* might be a

valid scope, but A.B, A*B, and A*B* would not. Under scoping, when a user provides an object spec, it is substituted for the variable part of the scope to form a complete pathname which is then subject to the standard NSW catalog lookup procedures. Naming an object relative to a scope is much the same as directory relative naming common on many systems. There is a single collection of scopes for each user which supports all

lookup operations regardless of the NSW operation being

¹ performed. The collection of scopes in effect at any time cannot be empty, but can refer to more than one region. Active scopes

² for a user session are initialized from a permanent node record. They can be modified either permanently in the node record or temporarily in the session record. A scope can be set only to a region of NSW name space to which the user has a lookup key (either private or public). All NSW object specs are looked up using scoping rules unless instructed not to by means of an explicit "unscope" request. Such a request is made by beginning the spec with the special symbol "\$", indicating that the name is to be looked up in the context of the entire catalog. The convention to be used throughout this document and throughout the NSW implementation 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 scopes in effect at the time. We sometimes refer to a name which employs scoping rules as a partial pathname specifier, whereas a name which does not employ scoping is a full pathname specifier.

Scope regions are specified using the same syntax as plural keys noted earlier. For example, \$ABC.DEF* denotes the scope covering the region of NSW object space with initial name components ABC.DEF. A scope cannot be incompletely specified (i.e. contain ellipses).

At any point in time the scoping set for a user can include:

¹ We could easily extend the design to support multiple collections of scopes which would be nameable, and dynamically selectable for referencing NSW objects. However, in order to keep things simple for the present, we refrain from doing so at this time.

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

- o a single scope region
- o a sequence of scope regions in which ordering is unimportant
- o a sequence of scope regions in which ordering is important

(b) is referred to as a parallel set of scopes, whereas (c) is a serial set of scopes.

Functionally, scoped lookup proceeds as follows:

- o If there is a single active scope region in effect, lookup the spec relative to the scope. This is accomplished by replacing the "*" of the scope with a "." and appending the spec to the scope to form the full name specifier. If the spec begins with a "\$", then the spec already represents a full name specifier and no further processing is necessary before catalog lookup. Catalog lookup proceeds as described in sections 4.1 and 4.2 for names without and with ellipses respectively.

example 1: if scope = \$A.B*

and object specifier was C.D, scoped lookup would be equivalent to lookup of \$A.B.C.D

example 2: if scope = \$A.B*

and object specifier was C.D.../type=file, then scoped lookup would be equivalent to lookup of \$A.B.C.D.../type=file

example 3: if scope = \$A.B*

and object specifier was \$C.D, the lookup would be unscoped searching for \$C.D

If there are parallel active scopes, lookup the supplied specifier as above relative to each of the scopes simultaneously. Create the set of objects which can be looked up with all of those full path specifiers. If no matching objects are found, lookup fails. If a single object is found, use it for satisfying the operation. If multiple matching objects are found, proceed as directed by user help to resolve to a single object. Then use it to satisfy the operation. If unable to resolve to a single object, lookup fails.

supplied object specifier relative to the first scope in the sequence,

- o If a single matching object results, the lookup succeeds and searching ceases.
- o If a multiple matches occur within the single scope, perform user disambiguation if provided; if disambiguation succeeds, the lookup succeeds and searching ceases; if disambiguation fails, the lookup operation fails.
- o If no object matches occur within the single scope, get next scope and repeat the above steps.
- o If no more scopes in the sequence, lookup fails.

A single active scope is somewhat equivalent to the concept of a working directory common to most systems. Series scopes are a slight generalization of search rule lookup, available on some systems. As specified here, NSW would support a single, unnamed scoping set, either parallel or serial. Obvious extensions would be to have more than one (named) scoping set, and or allow user specifiable combinations of serial and parallel searches.

5 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: scoped or unscoped names, with or without ellipses.

Ellipses in the name of an object being entered in the catalog serves to indicate that the name is to be "completed" in the context of the current catalog using the incomplete name lookup mechanism prescribed earlier. Scoped names with ellipses are resolved within the context of the current scopes only. Any incompletely specified name can only be used to replace an existing object (possibly create a new version if that were available). It can not be used to create a new object name.

The catalog enter function which determines the complete name for an entry spec is a slightly modified version of the NSW object lookup just described, and works as follows:

Find a set of candidate complete full path object names by

applying the basic NSW lookup mechanism to the entry spec.

If the resulting set contains a single object, the complete name of the existing object is used for the entry operation.

If the resulting set has no elements then: .

- o if the spec contained ellipses the entry operation fails
- o if the spec was already a full path specifier (i.e. beginning with "\$") use the spec as the complete entry name
- o if the spec required scoping and there was a single scope region, the complete name is the spec substituted for the variable part of the scope region
- o if the spec required scoping and multiple scope regions are being used in series, the complete name is the spec applied to the first scope region
- o if the spec required scoping and multiple scope regions are being used in parallel, the complete name is determined by user disambiguation among the entries for the spec applied to each scope region; else the operation fails
- o If the resulting set has multiple objects, the name is disambiguated by user help, if available.

An Enter right to the appropriate region is required as well as a Delete right if an object is being replaced, in addition to a lookup right. At the catalog lookup level, the type of the object can be required to match or not depending on whether the attribute "type" is included with the specifier or defaulted by the operation. Actual replacement of catalog objects can be confirmed or not based on parameters of the operation. For entering new objects into the catalog (as with accessing existing objects) the catalog object name phase may be followed by type and operation dependent access control checks before the operation can complete successfully.

[It will be important for users to understand that with serial scopes their first (or only) scope should include enter rights to allow the system to automatically create objects on their behalf (e.g. a temporary workspace name, a profile, etc.). Having the system automatically find a region to which the user has ENTER access seems like a bad idea. Another alternative might be to add the concept of a "login" region, as distinct from

AD-A185 967

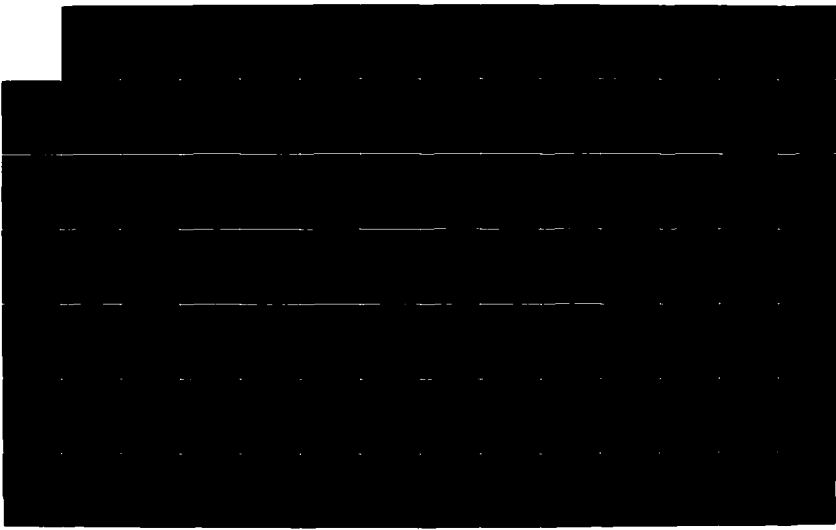
NSW (NATIONAL SOFTWARE WORKS) PERFORMANCE ENHANCEMENTS
(U) BOLT BERANEK AND NEWMAN INC CAMBRIDGE MA
R E SCHANTZ ET AL. JAN 81 BBN-4600

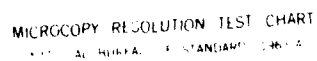
3/4

UNCLASSIFIED

F/G 12/5

NL





MICROCOPY RESOLUTION TEST CHART
 U.S. GOVERNMENT PRINTING OFFICE: 1963 O

a "connected" region which scoping represents. We do not plan to pursue this further at this time.]

NSW 6.0 will employ all of the preceding naming and lookup conventions for all catalog operations. In addition, existing user interfaces and system commands will be upgraded to reflect the modified operation of the catalog software.

System changes:

- o Modify WM catalog maintenance procedures to support new naming lookup and scoping conventions.
- o Modify (as necessary) ALL NSW components associated with the user interface to support the naming conventions when displaying NSW catalog names and regions.
- o Front End commands which must be modified are:

ALTER (scopes)	:	to support augmented scope types
SHOW SCOPES	:	to remove access type
SHOW FILES	:	to remove access type

- o Modify WM software to support public access regions.
- o Modify WM software to create private own space with node creation.
- o Add commands for permanently modifying scopes in the node record.

6 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 (a plural name), usually in conjunction with some form of processing common to the entire group. The SHOW OBJECTS command is perhaps the most prominent 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 plural name is equivalent to that of a partial name specifier using ellipses, with the exception that the special character "*" is used instead of the special character "...", to denote objects with zero or

more missing component names in place of the "*". We have already indicated an important special case of the plural naming convention when referencing key and scope regions. A region of NSW name space (key, or scope) is an unscoped plural name with exactly one "*" as its final component. A general plural name can have multiple "*" components, including preceding and trailing any specified name parts. "*" can appear only in the name part of a plural object specifier. "*" is implied for all unspecified attribute fields. The matching set can be reduced by including particular attribute values with the object specifier.

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. If the name in the example did not have a leading "\$", the lookup would be with respect to the scope setting prevailing at the time. Parallel scoping rules applied to a plural name generate a collection of objects which is the union of the plural name applied to each scope region. Serial scoping rules applied to a plural name generate the set of objects matching the name for the first scope which has a non-empty lookup result.

example:
Serial Scope in effect:

\$A.B*
\$A.C*
\$A.D*

Catalog contains objects {\$A.C.E, \$A.C.F.E, \$A.D.E, \$A.B.G}. The plural name *E would refer to the set {\$A.C.E, \$A.C.F.E}. Had the scopes been parallel, the appropriate set would be {\$A.C.E, \$A.C.F.E, \$A.D.E}. We defer further discussion of the use of plural names of this type within the context of NSW primitive operations.

NSW 6.0 will use and support plural naming conventions for all keys and scopes, and objects referenced with the SHOW command.

System changes:

- o Modify WM software to process the new form of plural

name for scopes, keys and object lookup.

- o Modify WM software and (if necessary) Front End software to accept and display plural names in the commands which accept or display keys, scopes and sets of objects (currently only SHOW).

7 ASPECTS OF THE FILE SYSTEM MODEL

NSW supports a copy model for workspace file processing. WS-CI's always make local copies of NSW files to be used for supporting a service session. In addition, unless otherwise requested by the accessing service, a tool reference to an NSW file will result in a nameable workspace copy of the file. 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. The NSW system does not maintain the mutual consistency of file copies. Optimizations can support read only access whereby the file is not actually copied when a locally accessible NSW file space image is already available; however the copy semantics must still prevail from the perspective of the accessing service. The copy semantics provides uniformity across translated (information lossy) and non-translated file movement, and across hosts which do not maintain local NSW file storage resources. A tool can specify a read only file access request, and to the extent enforceable or believable on the service host, NSW TBH software can make use of an optimized non-copy access path to the NSW file space image for satisfying the file accesses. The optimization has no effect on the NSW access control mechanism. [Comment: the workspace copy model serves as a mechanism for hiding some of the problem areas in the fundamental functionality of an NSW like file system and file system transfer capability; in particular, lack of host independent access methods, some information lossy file transfer modes, and automatic tool-specific file conversions make developing an integrated heterogeneous file system very difficult.]

7.1 FILE IMAGES

Users can request that an information loss-less image of an NSW file be MOVED from its current NSW storage host to another NSW storage host (if such a transfer is possible). 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 MOVE 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 (FBH), 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 as a result of tool oriented file movement are marked as a "system-directed-image." The retention of a cached image must be 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 (the FBH) manages the collection of system-directed images that it has decided to cache. A FBH 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 (WM) 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; updated file access protocols to support FBH file caching are given in a later section. 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 may select any image interchangeably. If a particular image is specified but unavailable, the operation will fail. Replacing or otherwise modifying a file catalog object invalidates all existing images of the file. The Delete operation defaults to all images of the file.

7.2 IMMOVABLE OBJECTS

When entering a file object in the NSW resource catalog (IMPORT, DELIVER), the object may be marked as "immovable." Existing files may also be subsequently so marked, according to appropriate access control and prevailing file state. An "immovable" file cannot be copied using NSW operations, nor can an image of it be made at other NSW file space supporting hosts. Such a file can be accessed only by services that run co-resident

on the file host, and only via direct access methods.

7.3 DIRECT FILE ACCESS

A service can request direct access to an NSW image of an NSW file. Such access can be granted only if:

- o the TBH can support this mode of access without violating the integrity of the NSW file system
- o a file image exists in local, NSW file space or can be moved to the TBH NSW file space (e.g. it is not marked immovable)
- o the file can be used by the service without any service specific transformations
- o appropriate catalog file locks can be set
- o user/tool has appropriate modify access rights

If any of these conditions are not met, the operation will fail. Granting direct access to an NSW file image immediately invalidates all other images of the file, and causes the image which is directly accessed to be marked as the original.

The NSW does not itself support the primitives for file access methods (i.e. for referencing internal file data). Direct file access is provided only through existing TBH specific file access methods where appropriate.

File catalog locks must be explicitly or implicitly released when the direct file access completes.

7.4 FILE SEMAPHORES [locks]

Users/services can request that a "lock" be set on an NSW file to control access to it as it undergoes modification. These file locks have historically within NSW been known as semaphores. 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:

- o exclusive lock, whereby only the locking user can access the file object in any way
- o 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)
- o 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 user 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.

File access requests have built-in lock requesting parameters. However, a file access need not also request that a lock be set, with the following exceptions. A request for direct write access to a file must include a request for an exclusive lock. A request for direct read access to a file must also include a request for either an exclusive or EWWC lock. The following table summarizes the behavior of the lock granting mechanism.

<u>current-lock-state</u>	<u>lock-request-type</u>	<u>result</u>
free	exclusive	exclusive,success
free	EWWC	EWWC,success
free	warning	warning,success
free	none	free,success
exclusive(X)	exclusive(Y)	exclusive(X),failure
exclusive	EWWC	exclusive,failure
exclusive	warning	exclusive,failure
exclusive	none	exclusive,failure
EWWC	exclusive	EWWC,failure
EWWC(X)	EWWC(Y)	EWWC(X),failure
EWWC	warning	EWWC,failure
EWWC	none	EWWC,success
warning	exclusive	warning,failure
warning	EWWC	warning,failure
warning(X)	warning(Y)	warning(X,Y),success
warning	none	warning,failure

When accessing an NSW file object, the operation proceeds in the following phases. First, the name is looked up in the catalog to identify the object. Then the appropriate object specific access control check is applied. Finally, and only after completing these initial phases, any requested locking specification is checked for consistency with the current lock state. If the use is consistent, the operation can be completed. If not, the operation fails.

NSW 6.0 will support read only file access, direct file access, immoveable objects, and full-file semaphore functionality. It will also support multiple image file management functions.

System changes:

- o Support protocol addition 10.2 and modify the GET and SEMAPHORE commands to allow augmented semaphore parameters, and other additional parameters.
- o Augment (if desirable) TBH software to make use of protocol additions in support of enhanced file access procedures.

8 NEW OBJECT TYPES

In addition to the object type "file" and object type "service" (extended from the NSW 4.1 type "tool") which are currently supported by NSW, device and workspace objects are also to be entered in the common NSW resource space. All object types can appear anywhere in NSW name space. The system implementation uses two regions \$SERVICES* and \$DEVICES* as repositories for generally accessible user services and line printer devices. Users may have their active scopes set to reference regions within \$SERVICES* and \$DEVICES* to facilitate naming these system supported services and devices.

However, because these two regions represent important system wide functions, we provide special purpose mechanisms to make the expected common access patterns easier without unduly burdening the single scoping mechanism, or requiring support for multiple named scopes. The first mechanism provides for conveniently looking up system wide service objects within one or more regions of \$SERVICES*. The mechanism is to have the system automatically append in serial scope fashion the relevant regions of \$SERVICES whenever a scoped request to instantiate a service fails to find a matching object. In effect, this merely causes the system to search designated system service areas when instantiating a service if it cannot be found in the user specified areas. It is the scope equivalent to public keys. In a similar fashion, operations which specifically refer to using device specs will search a public \$DEVICES* region, if lookup using user scopes fails.

The second mechanism provides for the session record containing an optionally specifiable default line printer device name which is initialized from the node record and can be used in conjunction with the PRINT file command. The default line printer would be modifiable temporarily or permanently like scopes. Since we are unlikely to support anything beyond lineprinters, and since one typically accesses only a single line printer, this mechanism should be sufficient for quite some time.

Permanent workspaces (when supported) are cataloged anywhere in user accessible NSW space at the time of registration. Temporary workspaces are automatically cataloged by the system, under a designated name entered in the appropriate region according to the user's scope setting in effect at the time the workspace is assigned. When externally referencing a file within a workspace, the NSW set file notation is used i.e. WORKSPACE-SPEC!WORKSPACE-FILE-NAME, where WORKSPACE is looked up under normal NSW name lookup conventions, and WORKSPACE-FILE-NAME is interpreted within the host workspace context.

When interacting with a service within a workspace, the naming context is automatically set to be the service workspace itself. There are two mechanisms for escaping from this implied "scope":

- o An NSW wide convention for specifying names relative to the NSW context (where the NSW scoping mechanism previously discussed is in effect). The convention is the use of a leading special symbol ("^") when entering an object name. This feature may not be supported for all interactive services on all TBH's.
- o Optional host specific escapes to reference non-workspace objects on the native host e.g. <directory> as a workspace escape for native mode on TOPS-20.

NSW 6.0 will support a system wide service region to be used when instantiating NSW services. This version will also support a standard "unscope from workspace" naming convention. The types of services available through NSW will be extended to include "native mode" and "ICP mode" services in addition to the current "friendly" interactive and batch services. Supported devices will be temporarily limited to line printers directly connected to UNIX-FE hosts, or TOPS-20 FE hosts, as devices private to their implementation.

System changes:

- o Add \$services public region to lookup for the USE command in the WM.
- o Modify interactive NSW mode TBH software to recognize and handle "escape to NSW" convention.
- o Add TYPE and LIST commands to FE, defaulting always to users TTY and local LPT (if any); requires protocol change 10.2; see also section 18 on UNIX-FE file handling capabilities for further detail.
- o Add TBH support for extended service types, including Workspace Command Interpreters (WS-CI).
- o Modify FE and WM tool initiation interaction to support ICP interface.

9 OBJECT TYPE-SPECIFIC ASPECTS OF THE RESOURCE CATALOG

Although all object types are entered into a common name space, and are referenceable via a common mechanism, there are object type specific operations, permissions, etc., which have meaning only when referencing an appropriately typed object within an appropriate operation. (In some cases, operations and permissions may be defined over more than one object type). In this section we attempt to enumerate a minimal set of operations and permissions for the anticipated NSW object types.

Note: the lookup, enter and delete permissions refer to the name space itself, regardless of object type.

For initial implementation, permissions to create and delete specific object types are subsumed by general catalog enter and delete permissions. Further control is provided by type dependent permissions governing the use of the cataloged objects.

Object Type = File

Appropriate Permissions:

COPY: access required to read a file and/or make a copy of it.

UPDATE: access required to change the contents of an NSW file object (either through replacement, or direct read/write access.

Appropriate Operations:

(these operations default their object type attribute to "file").

COPY
RENAME FILE
DELETE FILE
GET/DELIVER
LOOKUP FILE
ENTER FILE
IMPORT/EXPORT
SHOW FILES
SHOW DESCRIPTOR OF FILES
MOVE
...

Object Type = Service

Appropriate Permission:

EXECUTE: access required to instantiate the service

Appropriate Operations:

(These operations default their object type attribute to "service").

USE
SHOW SERVICES
SHOW DESCRIPTOR OF SERVICES
LOOKUP SERVICE
ENTER SERVICE
RENAME SERVICE
DELETE SERVICE

OBJECT TYPE = device

Appropriate Permission:

APPEND: access required to print a file

Appropriate Operations:

(These operations default the object type attribute to "device").

TYPE
PRINT
SHOW DEVICES
SHOW DESCRIPTOR of DEVICES

OBJECT TYPE = WORKSPACE

Appropriate Permission:

ACTIVATE: access required to instantiate a service in a designated workspace. A temporary permission for the workspace is added to the node record for each NSW workspaces assigned to the user as a result of invoking a service.

Appropriate file permissions govern access to the workspace

files for external manipulation via set file notation.

Appropriate Operations:

(These operations default the object type attribute to "workspace").

SHOW WORKSPACES
SHOW DESCRIPTOR OF WORKSPACES
ACTIVATE

OBJECT INDEPENDENT OPERATIONS:

Some operations can be invoked uniformly across all object types. These operations do not default the type attribute, and include:

SHOW OBJECT
SHOW DESCRIPTOR OF OBJECT
DELETE OBJECT
RENAME OBJECT

NSW 6.0 will include full support for object types file and service. It will also include support for object independent operations as they pertain to files and services. Generalized workspace and device object support will be deferred until Release 7.0.

System changes:

- o Add object descriptors for service objects.
- o Add a set of commands which support services in a manner similar to file, e.g., SHOW services; Rename services, etc. (in addition to existing USE command).
- o Add/modify support for keys of TYPE COPY, Modify (files) and Execute (services), including in ASSIGN-RIGHTS.
- o Add commands to support SHOW Descriptors of files and SHOW Descriptors of services.
- o Add command to support Register Service object.

10 FILE ACCESS and IMAGE MAINTENANCE PROTOCOL ENHANCEMENTS

The following three new WM calls are intended to support decentralization of the file access and file catalog update procedures as a means of improving NSW file access performance and allowing TBH implementations to control file image cacheing. The primitives are to be used to support the inclusion of file access capabilities in Foreman components, and to allow FE's access to the NSW file system. The implementation approach illustrates an architectural trend whereby the WM is used only for catalog lookup and access control, and the object manipulation is moved closer to the requesting agent. The addition of this functionality is completely backward compatible with the current file access protocols. The form of each primitive is identical, grouping the parameters into process identification data, exception handling data, logical object data, and (if appropriate) physical object data.

10.1 Enter-File-Object

This function ACCEPTS as arguments:

Identification

- o session id, or
- o service id, or
- o {user name + password strings}

Exception Handling

- o Help process
- o Disambiguation control parameter
- o Confirmation control parameter (what to do if there already is an object by this name i.e. use help confirmation);

Logical

- o file specifier (subject to complete name lookup rules; fully or partially specified;)
- o Attribute list for the new object; some attributes cannot be set from this list (e.g. time of creation) and are automatically inserted by the WM; attributes include possible marking as immovable object; attribute type=file is implicit with the primitive;
- o Lookup hint

Physical

- o physical file descriptor (for the new object in NSW file space);

gft

size of file (#bytes + byte size)
This function has RETURN arguments:

Confirmation

- o success/failure

Logical

- o Full path name object was put away as
- o Scope (region) which was used to create new entry (if any)
- o Lookup hint

NOTES:

The Enter-file-Object primitive is intended to support the catalog update part of the DELIVER (IMPORT) operation by processes within the "security Kernel." Attribute type=file is implicit with the primitive. Requires enter access + update access if actually replacing a file. "Identification" will

generally be service id. Session id and {user + password} are included for uniformity and completeness.

gft and gfd are copied from existing WM/FP primitives and could possibly be subsumable by an attribute.

"help process" is one of:

- o The calling process
- o Process designated as controller from the session record (usually the FE)
- o An arbitrary process id
- o No help process

Disambiguation control and confirmation control are parameters for allowing the calling process to control the use of help messages for ambiguous file references and operation confirmation. Both the full pathname of the object created and the scope (if any) which was actually used to create the full pathname are returned to support flexible service use of these individual name parts.

LOOKING AHEAD

At some point in the near future we are likely to try to support a modified form of the DELIVER scenario in which a component on a non-File Bearing host (e.g. UNIX FE) can directly provide a copy of a local file for inclusion in the NSW file system. We anticipate further modification to the Enter-file-object primitive to support this capability in the form of an extended PCD parameter. There seem to be at least two possible approaches. One is to have the PCD refer to a connection ID on which the donating process (i.e. the FE) is willing to send the appropriate file when requested by a SEND-ME message from a FP selected by the WM. Another is to have the FE component first interact with an FP on a FBH (which FBH is an issue here), and then have the PCD returned with the Enter-File-Object refer to the FBH copy. We defer further design on this topic for now.

10.2 Access-to-File-Object

This function ACCEPTS as arguments:

Identification

- o session id; or service id; or {user + password};

Exception Handling

- o Help process
- o Disambiguation control parameter (including how to handle lookup failure)
- o Confirmation control parameter

Logical

- o File spec (subject to general lookup rules);
- o Attribute list (those attributes, if any, which are to be used as disambiguation; attribute type = file is implicit with the primitive);
- o Access type (copy, direct);
- o Locking data (type, duration);
- o Lookup hint (optional; valid only with full file name specifier; see note below;

This function RETURNS as results:

Confirmation

- o Success/failure

Logical

- o Full NSW file name of object looked up;
- o Scope employed to locate object (if any);
- o Attribute list including creation timestamp, or version

- (if we ever support that);
- o New file specifier (if any was obtained);
- o Lookup hint (optional; can be used with any future reference to the file object; see below);

Physical

- o Physical file copy list (including a marked original)
gft gfd size of file (#bytes + byte size)

NOTES:

The Access-to-File-object primitive is intended to support the catalog lookup and access control check part of a file access procedure. It is intended that the FE or FM process use the returned object descriptors to interact directly with the appropriate FP to obtain a copy of the file (or access a local copy if accessible). When invoked from a FE, identification will typically be session id; from a FM, identification will typically be service id; alternatively, {user name + password} strings can be supplied as authentication data. In this case, a pseudo user session, lasting for the duration of the operation is created. If password is omitted, it can be obtained via user help, as specified. The file specifier and any listed attributes are used as augments to the lookup function. If a single object results, the access is checked based on the access type requested. If the accessing agent has the appropriate permission, the locking data is checked for consistency with current usage. If {user + password} authentication is supplied, only indefinite duration locks can be set. A failure at any of these steps results in a failure of the operation except for the case where lookup results in an empty set and the caller has requested via the disambiguation control parameter that the system gather an alternative file specifier (as requested by UCLA). In conjunction with this new primitive, each FP on every FBH must be modified to accept file movement requests from non-WM kernel components (if they do not already do so).

When this type of request is used by non-kernel (FE) processes, additional authentication messages may be required to achieve a secure system. Alternatively, it seems possible to develop a scheme whereby for non-kernel processes the Access-to-File-Object returns an encrypted capability which can be presented to and validated by the appropriate FP for retrieving the file. This approach would not change the pattern of communication for non-kernel processes, but would introduce

additional authentication overhead in all of the components. We defer further consideration of system security at this time in order to introduce new functionality without requiring system wide changes.

LOOKUP HINTS

The lookup hint is a means whereby the WM can provide an internal handle for a catalog entry in the WM data base to another NSW component when an object has been looked up in the catalog. The form of the hint might be an internal catalog identifier or virtual memory address. The idea is that when and if that object is subsequently subject to another lookup operation, the hint may provide a more efficient path to the appropriate catalog entry than would be the name specifier. Properties of the hint mechanism are that it is never required, and the lookup will logically yield the same result whether or not the hint was used.

The motivation for including a lookup hint type of parameter at this time is to solve the following problem related to supporting autonomous File Bearing Host (FBH) based file image management. By separating the file name lookup transaction from the physical copy manipulation transaction, we have also effectively removed the WM from the reply loop for indicating that an additional NSW file image is being retained at the FBH (in the case of an inter-host file reference). When the FBH software decides to cache a file image after an inter-host transfer, the physical descriptor data must be reported back to the WM for inclusion in the catalog so the image can be used to service subsequent references to the file.

There seem to be two ways to do this:

- o Conditionally including two additional specifically addressed messages (cached image data and confirmation) with the definition of the access-to-file transaction; or
- o Defining an independent file-image-update transaction which can be generically initiated when and if the cached image is retained;

The problem with the first approach is related to the decentralization of the caching decision. Since the decision is made by the FBH component, it must alert the WM of its intentions to cache an image along with the request for lookup so that the WM will continue the transaction beyond the reply to the lookup. This is made a little tricky since the desire to cache is

predicated on there being a transfer to begin with. That decision lies with the referencing component and may not be made until it views the PCD list. We reject as too error prone an extended transaction based on characteristics of the FBH caching implementation matching against the PCD list returned to it. A scheme whereby the WM always waits some time out interval for a possible additional reply is also feasible, but would seem to be adversely effected by high load situations under which caching might be all the more important.

The problem with adding an independent file-image management transaction seems to be the high overhead of recreating the context for referencing the file in question at the catalog. One important use of a continuing specifically addressed transaction is that the context associated with handling the particular operation is maintained within a (WM) process state or "hot" data base entry, and need not be recreated with each phase of the activity. Generic message sending does not seem to be significantly more expensive than specific message sending. Also, the current inter-message processing interval is very large relative to internal virtual memory management intervals, so the appropriate pages and memory maps will likely be swapped independent of whether it is handled by a single continuing process or an independent process. The major impact of an independent transaction is expected to be the replication of the name to object lookup for processing the independent parts of the "logical" transaction. To alleviate this overhead, we introduce lookup hints as arguments returned when an object is looked up. That hint can then be optionally supplied as a subsequent transaction parameter to optimize access to the same object.

Some proposed details of lookup hints for the catalog object update operation:

- o The hint itself is uninterpreted by the component which receives it; it is merely remembered and returned when the same object is to be referenced.
- o When using a hint, only a full complete pathname is acceptable (i.e. including \$ and no ellipses) for the file specifier.
- o There are two somewhat distinct uses of hints which can be supported. In one case, the reference is to any version of the object in the catalog under the complete name. In this case, no further name qualifier is required. A second case is where only a particular version of a catalog object can satisfy the request (e.g. for reporting a new image of an existing version). To support this second case, there must additionally be

some parameter by which the WM can validate that the object currently in the catalog under that name is the one to which the hint refers. Version numbers would appear to be suitable here, if they were part of our system. In their absence, we suggest using the object last modified timestamp attribute (already maintained in the catalog) as the validating parameter. This would mean that whenever lookup hints are given out, the last modified timestamp must be returned as an object attribute. When a lookup hint is used and intended to identify only a specific version of an object, the object specifier must include this attribute. If the lookup hint fails to refer to the object identified by the spec and any included attributes, the hint is ignored. If the catalog no longer contains an object with that name and those attributes (e.g. the version has been replaced), the operation fails. It is intended that the file-image-update operation include the last modified (version) attribute, and that the operation fail if the object it is updating is not the one to which the hint applies. This is what would normally happen if the name and creation attribute did not match the current catalog entry under that name, even without the hint.

- o The same hint and name specifiers would be usable for subsequent FBH to WM catalog updates to indicate that a cached image is to be deleted.

10.3 File-Image-Update

The primitive for initiating the transaction to add or remove cached file images from the catalog data base (used by "security kernel" processes) is File-Image-Update.

This function ACCEPTS as arguments:

Identification

- o Service id;

Exception Handling

None

Logical

Full file name specifier (only);

- o Attribute list (must include creation timestamp);
- o Lookup hint (optional, from a previous reference to the file object);
- o Type (add or remove a cached image PCD);

Physical

- o PCD to be added or removed;
- o size of file (#bytes + byte size)

This function RETURNS as results:

Confirmation

- o Keep it, or discard it (default on error condition or timeout is to retain with the option of retrying a removal at some future time);

NOTES:

The File-Image-Update primitive is intended to support the coordination of independent file cache management by FBH hosts with the maintenance of a central catalog of physical file images. It is anticipated that the addition of images to the cache would be handled by the component coordinating the individual inter-host file transfer (e.g. FM or FP), whereas removal might be the responsibility of a background file space management daemon process which periodically purges unreferenced images.

These are the primitives that NSW host components must use to support read only file access, direct file access, system directed file cacheing, optimized NSW file copying for service support, and other file movement related aspects of Release 6.0 functionality.

System changes:

- o Add these new primitives to WM.
- o Modify TBH/FE software to use the new primitives for interfacing to tool file requests/delivery, and FE file manipulation commands.
- o Augment WM-GET, WM-DELIVER to include optional additional parameters needed for controlling file system modifications.

11 AVOIDING TIMEOUT ON LONG FILE TRANSFER

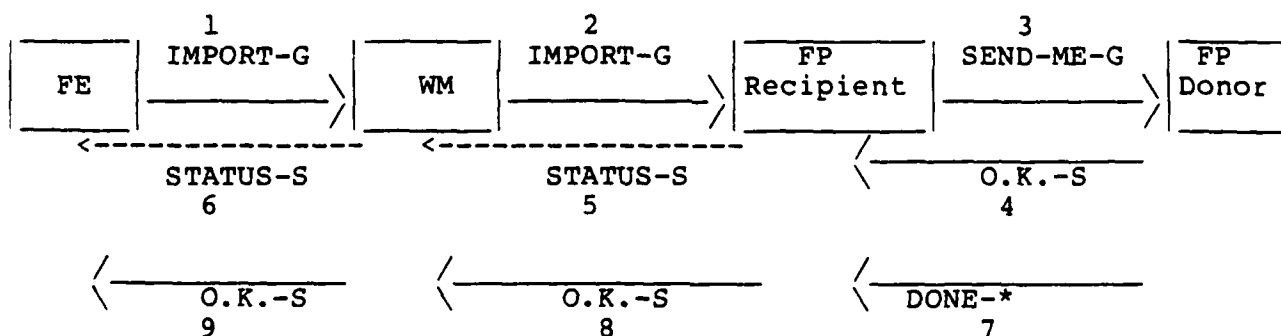
11.1 Intermediate Status Replies

To avoid possible transaction timeout by initiating or intermediary processes due to an unexpectedly long event, such as transferring a large file under heavy load, we introduce the concept of a partial reply for long transactions. Transactions are designated as short, or long. Long transactions are those that may have a large variance beyond the expected variance of message transmission and simple processing. Currently, only interhost file transfer will be defined to be a long transaction, although others may be so designated as the need is identified. For long transactions, we break the replies 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 is the normal termination of the transaction as currently supported. (See section 10.4 for a discussion of status querying during the second phase of a transaction). The responsibility for initiating the first phase status reply lies with the last process invoked in any designated long transaction chain. This process is identified in the documentation for each relevant NSW scenario. Status replies are sent to the process designated in the invoking message as the one to which completion replies are to be sent. This is a process which is presumably timing out a 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 back to any process which may be awaiting its final response. (In some cases, a message serving as a status reply may already be part of the scenario). The issue in selecting and

limiting those transactions which require intermediate status replies is one of avoiding the additional overhead in those cases where the status reply mechanism is of the same order of processing magnitude as the operation it is reporting. In general, one or more intermediate status replies should be acceptable (but not required) for any transaction using NSWTP conventions. For NSW version 6.0 we will specify and require its use in only the cases of file transfer to remedy the obvious system deficiency.

The status reply protocol applied to the case of NSW network file transfer would result in the following upgraded transfer scenario:



Messages marked as "-G" are generically addressed. Messages marked as "-S" are specifically addressed. The message marked as "-*" is not currently an MSG message at all but rather an explicit data transmission over the direct connection between the FP processes to support the file transfer. It is an optimization of sorts, to incorporate existing signals which serve other functions as well (messages 4 and 7) into the general transaction protocol sequence.

By adding an intermediary status reply which cascades over all of the processes of the transaction, we accomplish two things (a) we provide some relatively immediate feedback that the operation can be initiated and will proceed subject to loading factors on the hosts and the network (b) it provides initiating processes with the specific address of their generically addressed counterpart; this specific address can then be used to initiate further status queries, as discussed in section 12.

11.2 Handling Timeouts

Associated with the two phases of response for long transactions are two sets of timeout intervals. The short timeout interval (the one generally in use now) 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 operations to complete under variable load and size factors. A component would commit to the larger timeout because it has the prior knowledge that the operation has been successfully initiated, and the knowledge of the relevant operational parameters (e.g. size of file).

In general, a component in direct contact with the user can set very liberal timeouts provided the user is not locked out during the waiting period. If the user is prevented from any other parallel activity and from forcing the timeout to occur (i.e. aborting the operation) a shorter timeout interval, along with user help for renewing the timeout wait, seems advisable. A more complete solution to the timeout problem might include lower level guarantees of expedient delivery of timing signals. We will not pursue such an approach for NSW at this juncture. Rather, we will augment the intermediate response scheme with a general user oriented facility which allows one process to "poke" another process for its current status, given it has a specific address for the process. We will require that a long timeout be renewable if the communicating process responds to a status probe sent on expiration of the timeout period. This response will indicate that the component is still working on the request and that timeout may be premature.

NSW 6.0 will support the modified protocol which includes intermediate replies at the minimum for all scenarios which have an NSW inter-host file transfer operation as a subscenario. This includes the IMPORT, EXPORT, GET, DELIVER, COPY. The FP processes are responsible for initiating the intermediate response whenever a network file transfer is invoked. WM processes are responsible for relaying an intermediary response back to the original caller (FE or FM). In the case of direct FM/FE involvement with file transfer, there will not be an intermediary WM, and the FM/FE will be the direct recipient of the intermediate reply. Processes receiving an intermediate reply must remember the process name associated with the reply (identifying the next process in the chain), and must attempt and be unsuccessful in a direct status probe (see next section) of that next process before completing a self-initiated transaction timeout. User interface software will be modified to confirm self-initiated transaction timeout.

System changes:

- o Augment FP's on all TBH's to send intermediate responses.
- o Modify WM to record intermediate response parameters and pass on to original caller.
- o Modify FE/FM/WM processes to initiate direct status probe whenever self-initiated timeout of transaction occurs, and the specific name of the process working on the transaction is known.

A complete specification for intermediate status responses follows.

11.3 Intermediate Status Reply Specification

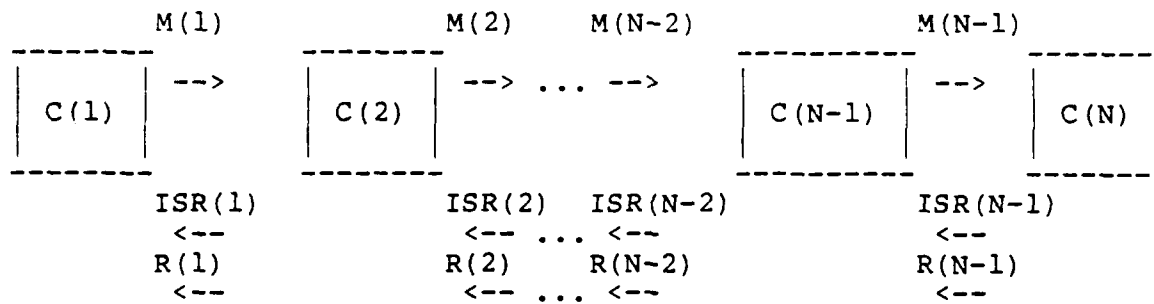
We define an Intermediate Status Reply (ISR) subscenario which may, in principle, be part of any NSW protocol scenario. We consider a model in which a chain of NSW components are involved in a scenario. For example, in a NSW network file transfer, the FE, WM, recipient FP, and donor FP form a chain of components of length 4. The ISR is initiated by the final component in the chain of components whenever:

1. the transaction requires the final component to open a direct network connection; or
2. the transaction performed by the final component is judged to take a "significant" amount of time; or
3. significant processing time has elapsed since the last ISR was sent.

Conditions (2) and (3) will not be further specified at this time.

All NSW components should be prepared to receive one or more ISR's following the initiation of any protocol scenario; and all components should be able to complete their protocol scenarios in the absence of ISR's.

Model:



The diagram above represents a chain of N components, in which 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 generic message $M(i)$ to it. Each message $M(i)$ has as an argument the transaction id $tid(i)$ generated by component $C(i)$. The maximum value of $tid(i)$ will be constrained by its possible subsequent use in a NSW status probe; this is discussed later. 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 $C(1)$.

Should the conditions for sending an ISR be met, component $C(N)$ will send one. The ISR neither invokes an operation (NSWTP type=1), is a reply to an operation (type=2), nor is a reply to an alarm (type=3). Therefore, we define a new NSWTP type value to be 'intermediate status reply', value = 4, which will take a form similar to a normal reply (type=2). The ISR sent by $C(N)$ will have structure:

4-List

```

{
  Index: type = 4                ;the new type
  Index: tid = tid(N-1)         ;the tid being serviced by C(N)
  0-List:                       ;always a null list
  1-List:                       ;result list of status
                                ;information which grows as
                                ;each component in chain adds
                                ;information
  {
    3-List:
    {
      BitStr: MSG process name of C(N)
      Index: tid(N-1)
      3-List:
      {
        Status List for C(N) (defined below)
      }
    }
  }
}

```

Each component in the chain between the final component and the initial must be able to receive an ISR, modify it (by adding information to the result list), and transmit it to the preceding component in the chain. When component C(i) receives an ISR, it will be of the form:

```

4-List:
{
    Index: type = 4
    Index: tid = tid(i)
    0-List:
    k-List:                                ;result list has k elements
    {
        3-List:
        {
            BitStr: MSG process name of C(N)
            Index: tid(N-1)
            3-List:
            {
                Status List for C(N)
            }
        }
        3-List:
        {
            BitStr: MSG process name of C(N-1)
            Index: tid(N-2)
            3-List:
            {
                Status List for C(N-1)
            }
        }
        .
        .
        .
        3-List:
        {
            BitStr: MSG process name of C(i+1)
            Index: tid(i)
            3-List:
            {
                Status List for C(i+1)
            }
        }
    }
}

```

Then, C(i) will attempt to match the tid of the ISR against a list of outstanding transactions for C(i). If the tid is not matched, C(i) will log the unmatched ISR in its error log file and then discard the message. If the tid is matched, C(i) will place tid(i-1) into the NSWTP header of the message, increment the result list count to k+1, and append

```
3-List:
{
    BitStr: MSG process name of C(i)
    Index: tid(i-1)
    3-List:
    {
        Status List for C(i)
    }
}
```

to the message. The resulting modified ISR is then transmitted to component C(i-1).

When component C(1) (usually a FE) receives an ISR, it will match the tid (the second argument) against a list of its outstanding tid's. If the match succeeds, C(1) will parse the rest of the message and use the information as desired. If the match fails, the message is logged as an error and then discarded.

Component C(N) will always send any ISR's before the final Reply message R(N-1) is sent. If a Reply message reaches component C(i) before the associated ISR, C(i) will process the Reply normally. The subsequently arriving ISR will fail to match any outstanding tid in C(i)'s list and will be discarded.

11.4 Status List Specification

The status list consists of:

1. Index: status code (major)
2. Index: status code (minor)
3. CharStr: specific information

Each component in the chain must insert values into the major and minor status code fields. Any component may insert a non-null string into its specific information field, but only component C(N) is required to do so.

The major status code indicates broadly the status of the component, and the minor status code gives more specific information within the broad category. The minor status code is a refinement of the major status code, and its value should explain or give a detailed reason for the value of the major

status code. The major code for 'Requested operation has not been initiated' followed by the minor code for 'NSW lookup failure' is illustrative.

We adopt the convention for both major and minor status codes that code values having low-order (decimal) digits in the range 00-49 refer to a success condition, and low-order (decimal) digits in the range 50-99 refer to a failure condition.

Possible values for the major status code are:

- 0 - no major status given (success presumed)
- 1 - running normally
- 2 - waiting (for message from other NSW component)
- 3 - requested operation has been initiated
- 4 - requested operation completed
- 51 - requested operation has not been initiated
- 52 - unable to complete requested operation

We define values for the minor status code based on the success/failure convention described above and, additionally, the NSW component id of the component as follows: the high-order (decimal) digits of the code are the component's NSW component id (and the code is specific to that component), or if equal to 0, the code is general (i.e., not component-specific). Possible values of component-independent minor status codes are given below; component-dependent minor status codes will not, in general, be specified at this time. An example of two minor status codes for the FE is given below.

- 0 - no minor status given (success presumed)
- 5 - notification of completion sent
- 6 - notification of completion not sent
- 11 - requested operation more than 10% complete
- 12 - requested operation more than 20% complete
- 13 - requested operation more than 30% complete
- 14 - requested operation more than 40% complete
- 15 - requested operation more than 50% complete
- 16 - requested operation more than 60% complete
- 17 - requested operation more than 70% complete
- 18 - requested operation more than 80% complete
- 19 - requested operation more than 90% complete
- 51 - primitive timed out
- 52 - operation unknown
- 53 - local host error
- 54 - network error
- 55 - NSW internal inconsistency
- 56 - NSW lookup failure
- 101 - running in Autologout
- 102 - running in Fastlogout

The major and minor codes may be combined in any way deemed meaningful for a given component, within the constraint that only success-related minor codes may be used with success-related major codes, and only failure-related minor with failure-related major codes.

The character strings inserted into the specific information field should be suitable for printing on the user's terminal and may not exceed 72 characters in length; each string should be printable in isolation, but should not include CR-LF at its end. The specific information field should convey information even more specific than that in the minor status field; however, it should not include MSG process names or be prefixed with an NSW component name (component names may be embedded in the string).

The FE will convert the major and minor status codes to character strings and will append the specific information field of the status list. The result is printed out on the user's terminal.

12 DISTRIBUTED "ARE YOU THERE"

12.1 Status Probes

To provide FE users with the ability to interrogate the status of long operations and on-going tool activity, and to allow NSW components to check on the status of pending operations as part of their timeout processing, we are specifying a status alarm to which all WM, FM and FP processes are expected to promptly reply with an indication of their current status.

We envision the user interface to a status probe facility allowing users to initiate "are you there" alarm signals directed at active service sessions and active long transactions which have already reported intermediary status. This would be in addition to the local "are you there" (^T) now supported directly within the FE process.

The reply to a status alarm serves two purposes:

(a) it serves to indicate that the appropriate processes are still active and (b) it provides a user text message indicating the current state of the transaction. Users may invoke "are you there" requests to verify that an on going long transaction is still progressing. System components may automatically invoke "are you there" signals (with a short response timeout) before initiating timeout recovery procedures when a long timeout is

about to expire.

Status replies which arrive after a final response to a transaction can be ignored. Status probes which refer to an unknown transaction return an indication of this condition.

NSW 6.0 will support a user initiated status probe against all active NSW service session and all active long transactions (i.e. those that have reported intermediate status). There will be FE commands/control sequences for initiating direct status probes, and for reporting the results of the probe to the user. All NSW components will invoke a status probe whenever a timeout for a long transaction is about to expire and an intermediate status response has been previously received.

System changes:

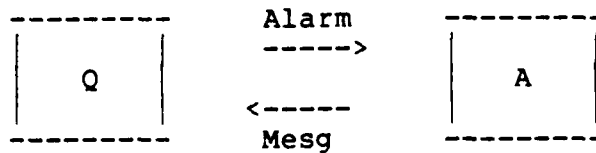
- o Add FE commands/control sequences for status probe initiation (direct and chained), and facility to display results (similar to help message handling).
- o Upgrade all NSW components (WM, FM, FP) to respond to and generate direct status probes as required.

A protocol specification for and user interface to the status probe facility follows.

12.2 Status Probe Specification

We define a NSW status probe subscenario which allows a component to "poke" another component with a request for status information, provided it knows the other process' specific address. The component receiving the status probe will respond by sending a long or short reply message to the component which initiated the probe. All NSW components must be responsive at all times to status probes. Since not all NSW components are currently responsive to unexpected specifically-addressed messages, the status probes are implemented as an MSG alarm, in which case every NSW component will be able to receive a probe at any time.

Model:



In the diagram above, component Q sends a status probe alarm to component A; component A responds with a specifically-addressed message to Q.

To obtain status from components which may be operating on more than one transaction at a time, it is necessary for the status probe to identify to A which transaction is being queried. The transaction is identified by sending to A the transaction id (known to A) under which the transaction was initiated; i.e., this is the tid under which A's activity was invoked. In general, since the probing component Q need not be the direct initiator of the original activity in A, the status probe reply should include a transaction id (known to Q) under which Q is handling the transaction. (An alternative approach would be to specify a global transaction id which can be used universally by all components associated with a protocol scenario. In the interest of minimizing extensive changes to existing components, we do not pursue this idea further.) Since we wish to be able to invoke various kinds of status probes and since alarms currently have functions already defined for them, the status probe must include a function code, which specifies how A should respond to the probe. Thus, the parameters necessarily associated with the status probe are:

1. destination address of A
2. id of the transaction for which status is sought
3. transaction id for tagging the response to Q
4. function code

Parameter (1) is the MSG process name of A. Parameters (2)-(4) must be encoded into the (16-bit) field defined for alarms.

A 2-bit function code is too restricted, although it would allow two 7-bit transaction id's to fit into the alarm field. A 4-bit function code allows 16 alarm functions to be invoked and

two 6-bit transaction id's. Sixteen alarm functions are adequate for the present and foreseeable future of NSW. A 6-bit transaction id allows values in the range 0 to 63 (0 as a tid has special meaning; see below) and should be adequate for all components since they are component-relative numbers. We adopt this allocation of the alarm code field and specify that, considering the bits from high-order to low-order, the status probe parameters be encoded as follows:

6-bits: tid(Q): tid generated by Q to handle the alarm
reply (value 1-63)
6-bits: tid(A): tid being serviced by A whose status is
being probed (value 0-63)
4-bits: function code (value 0-15)

Components are free to select transaction id's in any way, provided the range constraint is observed. Tid(A) is made known to all processes in a transaction chain using the result list of an ISR. If the value of tid(A) is not known (or there is no active transaction, as would be the case if a FE probed a FM associated with an active service instance) or if a component desires information about the global state of another component (e.g., all tid's being processed by a FE), then the value of tid(A) is zero. A component which receives a status probe whose destination tid in the alarm field (i.e., tid(A)) is zero should give global information about itself. The value of tid(Q) must also always be in the range 1 to 63. A tid value of 0 continues to have the meaning of 'no reply expected' as in current NSWTP; for a status probe to be meaningful, it should always request a reply.

All existing alarm codes retain their current values. To support a status probe which requests either a long or a short reply, we define function codes 15 and 14, respectively.

Both long and short reply messages will have type = 3 (reply to alarm), and the NSWTP tid field will be tid(Q). The third argument of the NSWTP header will be a null list if A is able to service the status probe (regardless of what the actual status returned in the fourth argument is), or an NSW error descriptor otherwise (e.g., the component does not recognize the alarm code function).

SHORT REPLY MESSAGE SPECIFICATION

The structure of a short reply to a status probe is:

```

4-List:
{
    Index: type = 3
    Index: tid = tid(Q)           ;from alarm code field
    NSWTP error descriptor list, or 0-List if no error:
                                   ;same as in current NSWTP
                                   ;Reply messages

    3-List:
    {
        Status List               ;as defined above for the ISR
    }
}

```

If tid(A) in the status probe was zero, the contents of the status list are to reflect the overall status of the component A. For components which service only one transaction at a time (e.g., WM or FP), such a status probe is equivalent to asking the status of A's active transaction. For components which service more than one transaction at a time (e.g., the UNIX FE) or which may not be actively processing a transaction (e.g., the FM), the status probe is equivalent to asking about A's overall status.

LONG REPLY MESSAGE SPECIFICATION

NSW release 6.0 will not support a long reply message to an alarm status probe. The contents of the long reply will be strongly component-dependent. A suggested possible long reply message structure for the FE follows:

```

4-List:
{
    Index: type = 3
    Index: tid = tid(Q)           ;from alarm code field
    NSWTP error descriptor list, or 0-list if no error:
                                   ;same as in current NSWTP
                                   ;reply messages

    3-List:
    {
        3-List:                   ;overall component status
        {
            Status List           ;as defined above for the ISR
        }
        k-List:                   ;list of information for tid's
                                   ;being serviced
        {
            3-List:
            {

```

```

        BitStr: MSG process name of initiator of tid(1)
        Index: tid(1)
        3-List:
        {
            Status List for tid(1)
        }
    }
    3-List:
    {
        BitStr: MSG process name of initiator of tid(2)
        Index: tid(2)
        3-List:
        {
            Status List for tid(2)
        }
    }
    .
    .
    .
    3-List:
    {
        BitStr: MSG process name of initiator of tid(k)
        Index: tid(k)
        3-List:
        {
            Status List for tid(k)
        }
    }
}
m-List:                                ;list of information for
{                                       ;associated services
    3-List:
    {
        BitStr: MSG process name of FM for service(1)
        Integer: service-id(1)
        3-List:
        {
            Status List for service(1)
        }
    }
    3-List:
    {
        BitStr: MSG process name of FM for service(2)
        Integer: service-id(2)
        3-List:
        {
            Status List for service(2)
        }
    }
}
.

```

```

      .
      .
      3-List:
      {
          BitStr: MSG process name of FM for service(m)
          Integer: service-id(m)
          3-List:
          {
              Status List for service(m)
          }
      }
  }
}

```

In this case, if the tid(A) field of the status probe is nonzero, then $k = 1$ and $m = 0$. That is, the list of tid information will include only the information for the tid specified as tid(A), and there will be no information on services at all. Ordinarily, for long replies it is expected that information for all tid's and services will be desired; thus, a null tid(A) field will be expected.

12.3 User Interface for Status Probing

The user interface for NSW 6.0 will provide a user with the ability to initiate status probes for both active commands and active services. Accordingly, at user command level, we define the commands:

```

SHOW      STATUS  COMMAND <command-number>
           SERVICE <connection-name>
           ALL

```

The SHOW STATUS COMMAND will initiate a probe requesting the status of the last component in the chain involved with processing FE command number <command-number> [the TOPS-20 FE will not support this command initially]. For those commands which did not induce an ISR, the FE will indicate this and display local status only.

We redefine the current control-T function slightly to accomodate the new status probe functionality. If control-T is issued when the user is at NSW command level (i.e., if the FE is ready to accept and act on user input), then its effect will be

to display status information found locally in the FE (such as return mode, whether logged in to NSW or not, NSW project name and node name if logged in, etc.). If, however, the user is waiting for a command to complete (in deferred return mode on the UNIX FE) or is in communication with a NSW service and types control-T, its effect will be to (1) print local FE status information relating only to the command or service which the user is waiting for or is in communication with, and (2) initiate a status probe to the NSW component associated with the command and service.

The user views the reply to a status probe in the same way as the output of any command, when operating in immediate return mode (on the UNIX FE). When operating in deferred return mode or communicating with a service, the reply to a status probe is displayed automatically to the user.

13 Specification of User Interface Changes

NSW 6.0 will contain a number of new user commands to support its increased and modified functionality. In this section, we describe the changes to the user interface; some existing commands (on NSW release 5) are retained and generalized, and new commands are added. Broadly, much of the changed user interface is organized around three commands: SHOW, SET, and MODIFY. There are additional changed commands in a fourth category, such as COPY, DELETE, etc., which reflect the augmentation of the catalog objects to which they may refer. The conceptual model underlying the SHOW-SET-MODIFY triple is that of the user (through the FE) querying or modifying the WM data bases which characterize the state of NSW objects, users, and user sessions.

The SHOW command allows the user to view the contents of many fields of the WM data bases to which user access is permitted; the SHOW command additionally supports access to data bases maintained locally by the FE in support of the user session. The SET command enables the user to insert new values into fields of WM data bases which are not privileged; the old contents of the fields are lost. Additionally, locally maintained data bases may be modified with the SET commands. The MODIFY command enables the user to modify fields of WM data bases which are not privileged. Modification is an incremental operation and is characterized in terms of adding, removing, or replacing particular values within a field.

The new user commands also reflect the fact that NSW 6.0 supports commands for both commonly designated typed objects

(namely, FILES and SERVICES, and in the future DEVICES and WORKSPACES) and commands for manipulating objects without a specific type designation.

13.1 Conventions

Changes to the user command interface are specified using the current NSW UNIX Front End as a base. In the following command descriptions, we use the following conventions:

1. A service is program or set of programs the user may invoke. The most significant class of service supported in NSW 6.0 is an NSW service and is the referent of many commands containing the string "SERVICE" or "SERVICES"; there are a few exceptions to this in which the term 'services' refers to all services, instead of just NSW services (see, e.g., SHOW ACTIVE SERVICES). An active service is a service that has been activated for use by the user; it may also be referred to as a service instance. On the UNIX FE, there are two additional classes of services supported, namely, TELNET connections to remote hosts and UNIX subshells running as inferior processes to the FE.
2. In the command syntax descriptions, optional arguments are enclosed in '[' and ']'; in the case of required argument fields, a choice between alternative values is indicated in the command syntax description by:

{ <choice1> | <choice2> | ... }

Or, the possible values an argument field may take may be indicated by:

<field-name> ::= <choice1> | <choice2> | ...

For example:

<retmode> ::= IMMEDIATE | DEFERRED

specifies that the <retmode> field may take one of the two possible values specified.

3. The character * (star) is used in the NSW object-naming syntax described above in Section 6. In addition, it

may appear in many argument fields where it will be interpreted by the FE as meaning 'all of' something. In many commands, 'ALL' is a synonym for '*' (when '*' is interpreted by the FE as the 'all of' specifier). Examples:

- a. 'SHOW FILES *' is a request to see the names of all files accessible by the user's node under the current scope. In this case, the '*' is a NSW file spec and is sent to the WM where it will be interpreted.
 - b. 'SHOW SEMAPHORES ALL ALL' is equivalent to 'SHOW SEMAPHORES * *', which is a request to see names of all files locked by the user's node. In this example, the FE interprets the 'ALL' or '*' as meaning all types and durations of semaphores; this information is encoded by the FE into the message sent to the WM.
4. The elements of a parameter list are separated by spaces (except in the commands referring to scopes, where distinguishing punctuation is used to denote different types of scopes; see below) and are surrounded by " (double quotes). A null parameter list is represented by "". If a parameter list contains one element or if it is the final parameter list of a command, the double quotes are not required. Examples:
- a. 'SHOW OBJECTS FORTRAN.*' is a request to display the names of the objects satisfying the spec FORTRAN.*. Note that the single element FORTRAN.* of the parameter list does not need to be surrounded by double quotes.
 - b. 'SHOW OBJECTS "A.B C.D.* WKSP...Ci"' is a request to display the names of objects satisfying the specs A.B, C.D.*, or WKSP...Ci.
 - c. 'MODIFY SCOPE REPLACE ALPHA.BETA ""' is a command to replace the scope region ALPHA.BETA with a null parameter list, i.e., remove the region from the user's scope.
5. The elements of a scope list are separated by + (plus) or by , (comma), to indicate that parallel and serial scope regions, respectively, are being referred to.

Within a scope list, white space (spaces or tabs) is ignored and may be included freely. Examples:

- a. `'SET SCOPE " A.B*, C.D*, E.F* "'` is a command to set the user's session scope to the (serial) regions indicated.
 - b. `'SET SCOPE "ALPHA.BETA*+GAMMA.DELTA*+EPSILON.THETA*"'` is a command to set the user's session scope the (parallel) regions indicated.
6. A command may be terminated without entering all its arguments, provided (1) the command is unambiguous to the point it has been entered, and (2) the remaining arguments have default values. The default values for commands terminated in this way are always displayed to the user and confirmed by the user before executing the command. Unless explicitly prohibited, any argument field may be null, i.e., take a default value, and we specify a default field value in each case. For `<connection-name>` fields, a null argument (default) is supported only if the FE has one active service; in this case, default is to that active service. Example:
 - a. Since ALL is the default for the arguments of the SHOW SEMAPHORE command, `'SHOW SEMAPHORES'`, `'SHOW SEMAPHORES ALL'`, and `'SHOW SEMAPHORES ALL ALL'` are equivalent commands.
7. The `<connection-name>` argument for active services (also called the `<service-instance-name>`) will be selectable by the user. This name may not be pure numeric, however, it may contain numeric characters. [This is to enable the UNIX FE to unambiguously support `'ABORT <command-#>'` commands; see ABORT below.] Example:
 - a. EDITOR-3 is a valid service instance name, but 33 is not.
8. Unless explicitly prohibited, for commands which support subcommands, multiple subcommands may be issued at a time. Example:

- a. The following is a SHOW NODE command requesting to see only the scope and permission fields of the node's WM data base record:

```
SHOW NODE,  
ONLY SCOPE,  
ONLY PERMISSIONS
```

9. If subcommands entered by the user conflict with the main command entered, the subcommands have higher precedence. See above in the SHOW NODE example: the default field list of the main command is ALL which would be what the user would see had he not specified subcommands; the subcommands override the ALL default, and the user will see only scope and permission data.
10. For commands which support field-selection subcommands, i.e., ONLY, EXCEPT, and ALL, any combination of these subcommands may be issued; the FE will compute the logical OR of the fields specified by the ONLY subcommands and the complement of those specified by EXCEPT subcommands. If the ALL subcommand is used, this will force all fields to be specified in the request. The resulting fields and their values will be displayed to the user.
11. The characteristics of the FE-user (i.e., the NSWExec) interface may be displayed and the representation character(s) of most control functions may be altered by the user. We adopt the convention that dot (.) in the <connection-name> field of the appropriate commands refers to the NSWExec interface. The representation of control functions of the FE-user interface which may be set on the UNIX FE are:

ABORT-INPUT (default: ^X)

RETYPE (default: ^R)

CHARACTER-DELETE (default: DEL)

WORD-DELETE (default: ^W)

QUOTE (default: ^V)

BEGIN-SUBCOMMAND (default: comma)

STATUS-QUERY (default: ^T)

SWITCH-EXEC (default: ^N)

the character the user types to switch from communicating with a service to communicating with the FE command interpreter.

EXEC-PROMPT (default: "NSW: ")

can be set to any string up to seven characters in length.

Refer to the UNIX FE User Manual for details concerning these functions.

12. Control characters entered as <control-char> arguments in commands dealing with CONTROL-FUNCTIONS may be entered in the following way:

- a. By typing '^', followed by the appropriate alphabetic; e.g., control-C is entered by typing '^C', and then 'C'.

13.2 SHOW Command

The SHOW command is used to query the (remote) WM data bases to which the user has access, or the (local) data bases maintained by the FE and associated with the user's NSW session. It is used to display to the user the current name and value(s) for system data base entries and various operational parameters. We discuss first those uses of the SHOW command which generate remote requests.

13.2.1 SHOW Commands which Initiate Remote Query Requests

The following commands display the names (and types, for generalized objects) of objects found by looking up the supplied specs in the NSW resource catalog:

SHOW OBJECTS <list of plural specs>
 ;displays names and types only

SHOW FILES <list of plural specs>
 ;displays names only

SHOW SERVICES <list of plural specs>
 ;displays names only

where:

plural specs are as defined above in Section 6 and may be scoped or unscoped. Default: all objects accessible by the node under the current scope. Examples:

1. 'SHOW SERVICES *.TECO.*' is a request to display the names of all NSW services within the user's scope whose names contain the string 'TECO'. 'SHOW PUBLIC SERVICES *.TECO.*' performs the same name lookup, but with reference to the "public" region.
2. 'SHOW FILES' makes use of the default and requests to see the names of all files accessible by the node under the current scope; it is equivalent to 'SHOW FILES *'.

Output:

Each spec and the objects it resolves to are listed together and distinguished from the other specs and objects, e.g.:

```
spec1
  object1 for spec1
  object2 for spec1
  .
  .
  .
  objectN for spec1
spec2
  object1 for spec2
  object2 for spec2
  .
  .
  .
  objectM for spec2
```

and so on.

The following commands display the values of the fields of the object descriptors of the objects found by looking up the given specs in the NSW resource catalog:

SHOW DESCRIPTOR (for) OBJECTS <list of plural specs>

SHOW DESCRIPTOR (for) FILES <list of plural specs>

SHOW DESCRIPTOR (for) SERVICES <list of plural specs>

where:

plural specs are as defined above in Section 6. Default: "*", all fields for all objects accessible by node under the current scope.

These commands support subcommands which enable field selection functions to be performed. They are:

ONLY <field-list>
 ;displays only the fields listed

EXCEPT <field-list>
 ;displays all fields except those listed

ALL
 ;displays all fields

<field-list> may not be null. The FE will recognize all field names that may be entered. Example:

1. The following requests the file size and host on which each file is stored of all files with BCP in their last name component:

```
SHOW DESCRIPTOR FILE *.BCP,  
ONLY SIZE,  
ONLY HOST
```

Output:

For each spec, the objects to which the spec resolves and fieldname/fieldname-values for each object are listed together, e.g.,:

```
spec1
  object1 for spec1
    fieldname1 value-of-fieldname1 for object1
    fieldname2 value-of-fieldname2 for object1
    .
    .
    .
    fieldnameN value-of-fieldnameN for object1
  object2 for spec1
    fieldname1 value-of-fieldname1 for object2
    fieldname2 value-of-fieldname2 for object2
    .
    .
    .
    fieldnameM value-of-fieldnameN for object2
  .
  .
  .

spec2
  object1 for spec2
    fieldname1 value-of-fieldname1 for object1
    fieldname2 value-of-fieldname2 for object1
    .
    .
    .
    fieldnameN value-of-fieldnameN for object1
  object2 for spec2
    fieldname1 value-of-fieldname1 for object2
    fieldname2 value-of-fieldname2 for object2
    .
    .
    .
    fieldnameN value-of-fieldnameN for object2
  .
  .
  .
```

and so on.

The following commands display the current session information regarding the scopes and permissions in effect for the user:

SHOW SCOPE ;displays current session scopes

SHOW PERMISSIONS ;displays current user permissions (keys)

The FE will display scope information to the user using the distinguishing punctuation defined for scopes (to show parallel and serial scopes).

Output:

An unstructured list of scope regions or permissions.

The following commands display current "public" information for scope, keys, and services:

```
SHOW PUBLIC (data for) SCOPE
                ;displays current "public" scope list
```

```
SHOW PUBLIC (data for) PERMISSIONS
                ;displays current "public" key list
```

```
SHOW PUBLIC (data for) SERVICES <service-spec list>
                ;displays names of service entries in public
                region, satisfying the service-specs in
                <service-spec list>
```

where:

<service-spec list> is a list of service specs using the syntax defined in Sections 1 and 6. Default: *, i.e., all services in the "public" region.

Output:

[For SCOPE and PERMISSIONS:] An unstructured list of scope regions or keys.

[For SERVICES:] A list of services in the public region such that associated with each service spec is a list of services to which the spec resolves, e.g.:

```
service-spec1
  servicel for spec1
  sercice2 for spec1
  .
  .
  .
  serviceN for spec1

service-spec2
  sercicel for spec2
  service2 for spec2
  .
  .
  .
  serviceM for spec2
```

and so on.

The following command displays names of files locked by this node, as selected by the arguments:

```
SHOW SEMAPHORES (of type) <type> (and duration) <duration>
```

where:

```
<type> ::= EXCLUSIVE-ACCESS | SINGLE-WRITER | ALL
```

```
<duration> ::= INDEFINITE | SESSION | ALL
```

In this command, ALL is a synonym for *. Default: ALL.
Examples:

1. 'SHOW SEMAPHORES SINGLE-WRITER SESSION' is a valid command and requests the names of all files locked by the user's node for the duration of the user session (SESSION) in SINGLE-WRITER mode.
2. 'SHOW SEMAPHORES SESSION' is an invalid use of default; the <type> argument must be represented explicitly in the command in some way, e.g., 'SHOW SEMAPHORES * SESSION'.

Output:

An unstructured list of files locked by node and satisfying the command's arguments.

The following commands display NSW system-wide information, as selected by their arguments:

SHOW SYSTEM HOSTS

 ;displays hosts of NSW configuration

SHOW SYSTEM USERS <project> <node>

 ;display information for each user logged in
 satisfying the arguments

where:

<project> ::= name of project | ALL | .

<node> ::= name of node | ALL | .

In this command, ALL is a synonym for *. A dot (.) in <project> and/or <node> means, respectively, current logged-in project and/or node. This command may be issued by a user not logged in (as SYSTAT on TENEX/TOPS20 systems); the dot notation is not allowed if the user is not logged in. Default: ALL.
Examples:

1. 'SHOW SYSTEM USERS . .' obtains system information about the user issuing the command.
2. 'SHOW SYSTEM USERS' obtains system information about all current users of NSW.

Output:

An unstructured list of hosts or project+nodes satisfying the arguments of the command.

The following commands display the local status (where meaningful) and initiate status probes to display remote status information to the user. Remote status probe transactions are initiated whenever the FE has the MSG specific address of the NSW process executing the command as based on an ISR [see Section 11.1] or the name of the corresponding Foreman component for the specified service.

SHOW STATUS (of) COMMAND <command #>
SHOW STATUS (of) SERVICE <service-instance-name>
SHOW STATUS (of) SEMAPHORE <list of filespecs>
SHOW STATUS (of) JOB <job #>

for COMMAND:

<command #> is number of command about which status information is desired; if null, defaults to oldest command outstanding. The effect of the command is to (1) display local FE information about the command, (2) send the status probe (if remote specific MSG address is known to FE), and (3) display result of probe when ready. [The TOPS-20 FE will not currently support the SHOW STATUS of COMMAND function.]

for SERVICE:

<service-instance-name> is the (user-assigned, or FE-assigned) name by which the user refers to the requested service. The effect of the command is to (1) display local FE status information about the service session, (2), send a status probe, and (3) display results of probe when ready.

for SEMAPHORE:

[replaces SEMAPHORE READ command in NSW release 5] Default: all files locked by the initiating node.

for JOB:

[replaces JOB command in NSW release 5] The <job #> is supplied to the user by the programs which accept the user's batch job request. Default: not allowed.

Examples:

1. 'SHOW STATUS COMMAND 6' requests information about command number 6; 'SHOW STATUS COMMAND' requests information about the oldest command outstanding.
2. 'SHOW STATUS JOB 6654' requests status information about batch job 6654, where this number is obtained as output from the batch job acceptor.

Output:

for COMMAND, SERVICE, and JOB: An unstructured string of status information.

for SEMAPHORE: a structured list giving, for each file spec, the files the spec resolves to and semaphore status information for each file, e.g.:

```
spec1
  file1 for spec1, status-info for file1
  file2 for spec1, status-info for file2
  .
  .
  .
  fileN for spec1, status-info for fileN

spec2
  file1 for spec2, status-info for file1
  file2 for spec2, status-info for file2
  .
  .
  .
  fileM for spec2, status-info for fileM
```

and so on.

The following commands display data base entries as selected by their arguments for the node record or the current session record for the initiating user:

```
SHOW NODE (data base) <field-list> | ALL
SHOW SESSION (data base) field-list> | ALL
```

where:

<field-list> is either a node-specific list of field names or a session-specific list of field names. ALL is a synonym for *; the default is ALL.

These commands support subcommands which enable more complicated field selection functions to be performed. They are:

```
ONLY <field-list>
      ;displays only the fields listed
```

EXCEPT <field-list>
 ;displays all fields except those listed

ALL
 ;displays all fields

Subcommand specifications have higher precedence than main command specifications. <field-list> in a subcommand may not be null.

To minimize the effect of user error and the resulting needless network traffic, the FE will recognize the set of possible field names that are acceptable. A partial listing of field names follows:

Node-specific: scope, printer, permissions, saved-sessions

Session-specific:
 scope, permissions, active-services

Note: SHOW SESSION SCOPE is equivalent to SHOW SCOPE. SHOW SESSION PERMISSIONS is equivalent to SHOW PERMISSIONS. The shortened versions of these commands are provided for user convenience. Examples:

1. 'SHOW NODE' is a request to display all information in the WM record; it is equivalent to 'SHOW NODE ALL' and to

SHOW NODE,
ALL

2. The use of differing subcommands is illustrated by:

SHOW SESSION,
ONLY SCOPE,
EXCEPT SERVICE

It is a request to display information from the current WM session record; the presence of subcommands as field selectors overrides the default 'ALL' in the main command. 'ONLY SCOPE' requests only the current session scope, but the presence of 'EXCEPT SERVICE' (which excludes service information) modifies the

request to be for all session information except service information.

3. 'SHOW NODE SCOPE SAVED-SESSIONS' illustrates the means of selecting fields by arguments in the main command. Note that the field list is not inside double quotes; that is not necessary since it is the terminating field list.

Output:

These commands display field names, followed by field values, in a manner that associates each field name with its values, e.g.:

```
field-name1
    value-list for field-name1
field-name2
    value-list for field-name2
```

and so on.

13.2.2 SHOW Commands which Do Not Initiate Remote Query Requests

This section describes SHOW commands which are satisfied by scanning (local) FE data bases and displaying the results to the user. They are:

SHOW ACTIVE COMMANDS

 ;[UNIX FE only] displays list of active commands
 and their status

SHOW ACTIVE SERVICES

 ;displays list of active services and their
 status

SHOW CONTROL-FUNCTIONS (for) <connection-name>

 ;displays control functions in effect for
 <connection-name>. For NSW services, displays
 NSW control functions; for TELNET connections and
 UNIX subshells, display terminal mode. Use of
 dot (.) for <connection-name> causes FE control
 characters and control functions to be displayed.

Output:

These commands display their output in a format which is suitable for their application; for SERVICES, this might be, e.g.:

Active services:

```
    servicel    status-info for servicel
    service2    status-info for service2
    .
    .
    .
    serviceN    status-info for serviceN
```

The SHOW FRONT-END command is used to display various FE operational parameters:

```
SHOW FRONT-END RETURN-MODE
                        ;[UNIX FE only] displays current FE return mode
```

```
SHOW FRONT-END TERMINAL-MODE
                        ;displays current FE terminal mode
```

```
SHOW FRONT-END STATUS
                        ;displays current FE return-mode [UNIX FE only],
                        date and time, whether user is logged in to NSW
                        or not (if logged in, display project and node),
                        whether LOGIN or LOGOUT command is pending or
                        not, # active services, and # active commands
                        [UNIX FE only].
```

Output:

These commands display their output in a format which is suitable for their application.

13.3 SET Command

The SET command enables the user to set the values of various WM and FE data base entries and operational parameters. The previous values are lost. The FE assumes no responsibility for inter-field consistency; that is, if some fields of a catalog entry are mutually dependent and the user does not set all of

them, the inconsistency will not be detected by the FE. As with the SHOW command, some SET commands (those dealing with WM data bases) generate remote activity, while others are handled exclusively within the FE.

The output displayed to the user is either a short note confirming that the requested operation was performed or (in the case of a local operation) possibly no output at all.

13.3.1 SET Commands which Generate Remote Activity

The following commands assign values to fields in WM data bases; one command invocation can handle multiple assignments. The most general case (multiple assignments of multiple-valued fields) is shown below:

```
SET NODE (data base) <field-name1> (to) <values1>
                                <field-name2> (to) <values2> ...
SET SESSION (data base) <field-name1> (to) <values1>
                                <field-name2> (to) <values2> ...

SET DESCRIPTOR (for) OBJECT <object-spec> (field name)
                                <field-name1> (to) <values1>
                                <field-name2> (to) <values2> ...
SET DESCRIPTOR (for) FILE <file-spec> (field name)
                                <field-name1> (to) <values1>
                                <field-name2> (to) <values2> ...
SET DESCRIPTOR (for) SERVICE <service-spec> (field name)
                                <field-name1> (to) <values1>
                                <field-name2> (to) <values2> ...
```

where:

The <valuesi> arguments are parameter lists surrounded by double quotes. A null parameter list is represented by "". The FE will automatically insert CR-LF into the user's terminal buffer at appropriate places, since these commands can easily exceed one line in length. Not all fields may be settable from the FE without special permission. As with SHOW NODE and SHOW SESSION, the FE will recognize the set of allowed field names. Defaults or * are not allowed. Example:

1. 'SET SESSION SCOPE "A.B*, C.D*" ' sets the node's session scope to the (serial) scope-regions in the quoted argument. [This is equivalent to 'SET SCOPE "A.B* , C.D*" '.]

The following command sets the current session scope to its argument. This does not have any effect on the scope setting in the node record. The SET SCOPE command is a shorthand for the SET SESSION SCOPE command. It is provided as a separate command for user convenience.

SET SCOPE (to) <scope-region list>

where:

<scope-region-list> is a list of scope regions separated by + (for parallel regions) or ',' (for serial regions). Default: set session scope to node's permanent scope region list. Refer to Section 4.3 for a discussion about scope. Example:

1. 'SET SCOPE' sets the session scope to the node's permanent scope region list; this list may be displayed by issuing 'SHOW NODE SCOPE'.

The following command sets the semaphore variable on the filespec given by its argument:

SET SEMAPHORE (on) <filespec> (to value) <type>
(for duration) <duration>

where:

<filespec> is a filespec which may contain a *

<type> ::= EXCLUSIVE-ACCESS | SINGLE-WRITER | FREE

<duration> ::= INDEFINITE | SESSION

Default: not allowed.

The following command sets the user's node password [replaces the PASSWORD command in NSW release 5]:

SET PASSWORD <paswd> <paswd>

where:

<passwd> is the new password (not echoed to user); the user must enter it twice as a reliability measure.

13.3.2 SET Commands which Do Not Generate Remote Activity

This section describes SET commands which set values in (local) FE data bases. They are:

SET FRONT-END RETURN-MODE (to be) <ret-mod>
;[UNIX FE only]

where:

<ret-mod> ::= IMMEDIATE | DEFERRED

Default: none allowed.

SET FRONT-END TERMINAL-MODE <param-name> [<arg>] ...

where:

<param-name> is a terminal mode parameter; if it takes an argument, <arg> follows, otherwise, another <param-name> may follow. Representative <param-name> and <arg> fields are:

C100
VT100
VT52
TTY
PAGE-LENGTH <number>
LINE-WIDTH <number>
FLOW-CONTROL {ON | OFF}

Example:

1. 'SET FRONT-END TERMINAL-MODE PAGE-LENGTH 23 LINE-WIDTH 66' sets the user terminal's page-length and line-width parameters.

The following command sets a control function for an active

NSW service or the user-FE interface; initially, we will support only NSW services:

```
SET CONTROL-FUNCTION (for) <connection-name> (for function)
    <control-fn> (to be) <control-char>
```

where:

<connection-name> is the name by which the user refers to the active service; <control-fn> is the name of a control function which may be set; and <control-char> is either an Ascii representation of a control character, i.e., '^' followed by some letter, or the octal character itself (i.e., user types the control character).

A dot (.) as <connection-name> refers to the user-FE interface and is the means by which the user changes the characteristics of that interface. Example:

1. 'SET CONTROL-FUNCTION 3QEDX-RM CHARACTER-DELETE DEL' requests the control function CHARACTER-DELETE on <connection-name> 3QEDX-RM to be handled by the DEL character, instead of the default (on the UNIX FE, the CHARACTER-DELETE default for Multics services is ^H). The FE recognizes 'DEL' in this command as referring to the Ascii character octal 177.

13.4 MODIFY Command

The MODIFY command enables the user to modify values of WM data base catalog entries. Any existing values not referenced by a MODIFY comand remain unchanged; the command allows values in the catalog entries to be added, removed, or replaced. Currently, all MODIFY commands generate remote activity.

The output displayed to the user is a short note indicating that the requested operation has been performed.

The following commands, respectively, modify node data, session data, and objects descriptors in the WM:

```
MODIFY NODE      <field-name>
                  ADD      <vlist>
                  REMOVE   <vlist>
                  REPLACE
                    <value1> (with) <vlist1>
                    <value2> (with) <vlist2> ...
```

```
MODIFY SESSION  <field-name>
                  ADD      <vlist>
                  REMOVE   <vlist>
                  REPLACE
                    <value1> (with) <vlist1>
                    <value2> (with) <vlist2> ...
```

(This command modifies the session scope only --
is synonymous with MODIFY SESSION SCOPE.)

```
MODIFY SCOPE    ADD      <vlist>
                  REMOVE   <vlist>
                  REPLACE
                    <value1> (with) <vlist1>
                    <value2> (with) <vlist2> ...
```

```
MODIFY DESCRIPTOR (for) OBJECT <object-spec> <field-name>
                  ADD      <vlist>
                  REMOVE   <vlist>
                  REPLACE
                    <value1> (with) <vlist1>
                    <value2> (with) <vlist2> ...
```

```
MODIFY DESCRIPTOR (for) FILE <file-spec> <field-name>
                  ADD      <vlist>
                  REMOVE   <vlist>
                  REPLACE
                    <value1> (with) <vlist1>
                    <value2> (with) <vlist2> ...
```

```
MODIFY DESCRIPTOR (for) SERVICE <service-spec> <field-name>
                  ADD      <vlist>
                  REMOVE   <vlist>
                  REPLACE
                    <value1> (with) <vlist1>
                    <value2> (with) <vlist2> ...
```

where:

<field-name> is a FE-recognized name of the field to be modified; <vlist> is a list of values to be added or removed and surrounded by double quotes; <vlistN> is a list of values to replace <valueN>. If <vlist> or <vlistN> contains more than one value and is not terminal, it must be surrounded by double quotes. Examples:

1. <<<to be supplied>>>

13.5 Other New Commands

13.5.1 DELETE, RENAME, and COPY Commands

The following commands manipulate files, services, and objects and display full names of objects manipulated as the result of the command (including site and type parameters, where appropriate); the output displayed to the user is a short note confirming that the requested operation was performed.

DELETE FILE <spec>

DELETE SERVICE <spec>

DELETE OBJECT <spec>

RENAME FILE <spec> (to be) <spec>

RENAME SERVICE <spec> (to be) <spec>

RENAME OBJECT <spec> (to be) <spec>

COPY FILE <spec> (to) <spec>

COPY SERVICE <spec> (to) <spec>

COPY OBJECT <spec> (to) <spec>

where:

<spec> is as defined above in Section 1.

13.5.2 PLACE Command

The following command places a file image at a given destination (see discussion of file image above in Section 7.1);

the output displayed to the user is a short note indicating that the operation was performed and a list of current file image sites.

PLACE (an image of file) <filespec> (at) <site>

where:

<site> is an ARPANET NSW host name or number.

This command accepts the subcommands

RETAIN ;do not delete image on origin host

DELETE ;default - delete file image on origin host

Output:

Disposition of operation, and (if successful) a list of sites of current images of the file.

13.5.3 TYPE, PRINT, and GET Commands

Refer to Section 18.1.

13.5.4 USE Command

The USE command is used to instantiate an NSW service, optionally specifying parameters to be passed to the service-instance and allowing the user to optionally provide the local (FE) name for the connection to the service. Additionally, tool kits are supported by allowing a single command to include multiple service specs (which must all be runnable on a single TBH). When multiple service specs are indicated, the TBH must support a workspace command interpreter (see Section 17) to coordinate the invocation of each of the individual services. The syntax is:

USE [<connection-name>=] <service-arg1> <service-arg2> ...

where:

one or more <service-arg> fields are acceptable, and each <service-arg> has the form:

```
<service-arg> ::=  
    <service-spec> | <service-spec> <param-list>
```

and <connection-name> is an optional user-assigned name under which the connection is referred to locally. If <connection-name> is omitted, the FE will generate a connection name for the service-instance.

The command supports subcommands to augment the functionality above and provide alternate syntax for it. The subcommands are:

```
NAME <connection-name>  
    ;if specified, overrides <connection-name> given  
    in main command
```

```
SERVICE <service-arg>  
    ;where <service-arg> is the same as above. It is  
    treated as if it were appended to the list of  
    <service-arg>s in the main command.
```

Examples:

1. 'USE EDITOR-3=TECO-IC' requests a Teco service from ISIC to be referred to as EDITOR-3. The following is equivalent:

```
USE TECO-IC,  
NAME EDITOR-3
```

In this case, the NAME subcommand overrides the default (null) specification in the main command.

2. 'USE TECO-IC BCPL-IC LOADER-IC' is a request to create a tool kit with a FE-supplied <connection-name> to include the services listed.
3. The following is equivalent in effect to that in the previous example:


```
USE,  
SERVICE TECO-IC,  
SERVICE BCPL-IC,  
SERVICE LOADER-IC
```

The FE transmits the service specs to the WM in the order they are entered by the user. This order could be significant to the TBH.

4. 'USE TECO-R2 "SIGNAL.FOR"' is an example of the use of a service parameter; the filename SIGNAL.FOR is passed to the Teco service as a parameter.

13.5.5 SAVE Command

The following command initiates an LND-MAINT transaction on the named connection (service instance); if accepted, the affected session will be RESUMEable at some future time. This is equivalent to detaching the NSW service session.

SAVE (workspace for) <connection-name>

13.5.6 ABORT and TERMINATE Commands

The ABORT and TERMINATE commands, respectively, replace the QUIT ABORT and QUIT TERMINATE commands of NSW release 5; on the UNIX FE, TERMINATE also replaces QUIT SHELL and QUIT TELNET. The syntax is:

```
ABORT {<connection-name> | <command-#>}  
      ;ends service session without saving workspace  
      files, or aborts command number <command-#> [UNIX  
      FE only].
```

```
TERMINATE <connection-name>  
      ;ends service session after saving workspace  
      files
```

13.5.7 LOGIN Command Interface Change

The Reply message to a successful login request will contain relevant parts of node record logged-in information, such as project name, node name, date/time of last successful login, and so on. A selected set of this information is displayed to the user.

13.6 Command Tree of Changed Commands

The following is a Command Tree of the changed commands,
with selected arguments included on some commands:

```

SHOW  OBJECTS <spec-list>
      FILES <spec-list>
      SERVICES <spec-list>
      DESCRIPTOR      OBJECTS <spec-list>
                      FILES <spec-list>
                      SERVICES <spec-list>

      NODE <field-list>
      SESSION <field-list>
      SCOPE
      PERMISSIONS
      PUBLIC          SCOPE
                      PERMISSIONS
                      SERVICES <spec-list>

      SEMAPHORES
      SYSTEM          HOSTS
                      USERS
      STATUS          COMMAND
                      SERVICE
                      SEMAPHORE
                      JOB
      ACTIVE          COMMANDS
                      SERVICES

      CONTROL-FUNCTIONS
      FRONT-END      RETURN-MODE
                      TERMINAL-MODE
                      STATUS

SET   NODE
      SESSION

      SCOPE
      DESCRIPTOR      OBJECT
                      FILE
                      SERVICE

      SEMAPHORE
      PASSWORD
      FRONT-END      RETURN-MODE
                      TERMINAL-MODE

      CONTROL-FUNCTION

```

MODIFY	NODE	
	SESSION	
	SCOPE	
	DESCRIPTOR	OBJECT
		FILE
		SERVICE

DELETE	OBJECT
	FILE
	SERVICE

RENAME	OBJECT
	FILE
	SERVICE

COPY	OBJECT
	FILE
	SERVICE

PLACE

USE

SAVE

ABORT

TERMINATE

14 Specification of Associated NSWTP Changes

To support the user interface changes described above, changes to NSWTP are required. Unlike NSW release 5, which utilized unstructured text as the fourth argument of the NSWTP reply message, NSW 6.0 will utilize a structured form (not text readable) in the fourth argument. The FE will convert the structured format to a human-readable form at display time.

The protocol changes reflect the new ability to query and modify WM data bases by field name as described above with the SHOW-SET-MODIFY triple of commands.

14.1 Conventions

In the protocol descriptions, we use the following conventions:

1. We define the NSWB8 Index quantity obj-typ to be:
 - a. 0 - object: any object meaningful in the command
 - b. 1 - file: refers to file object
 - c. 2 - service: refers to service object
 - d. 3 - device: refers to device object (not used for release 6.0)
2. We define the NSWB8 Boolean qdata to be:
 - a. TRUE: if following NSWB8 element is a List of values.
 - b. FALSE: if following NSWB8 element contains fieldname-qdata-values triples of an embedded data structure.
3. We define the NSWB8 Index quantity fld-op to be:
 - a. 1 - add: add following list to data base
 - b. 2 - remove: remove following list from data base
 - c. 3 - replace: replace following value in data base with following list
4. We define the NSWB8 Index quantity fld-sel to be:
 - a. 0 - all: return values for all fields
 - b. 1 - only: return values only for following list of fields
 - c. 2 - except: return values for all fields except

those in following list

5. We use the following message arguments as follows:

- a. id: (Index) WM-assigned user session id
- b. dstring: CHARSTR to be displayed to user at completion of operation
- c. hlpdsp: same as in release 5
- d. m-attr-list: Match Attribute List; refer to Section 4.2.2.29 in WM Subsystem Specification document. The FE will fill this list with appropriate defaults.
- e. s-attr-list: Source Attribute List; refer to Section 4.2.2.30 in WM Subsystem Specification document. The FE will fill this list with appropriate defaults.

6. The null list in a NSWTP type = 1 message is used to represent 'whatever is there'; thus, e.g., the following is a request to display all unprivileged information available for file A.B:

```
WM-SHOW-DESCRIPTOR(id,obj-typ=1,List("A.B"),  
                  fld-sel=0,List(0))
```

7. The null list in a NSWTP type = 2 message is used to represent 'no match'; e.g., if a reply message to a WM-SHOW-DESCRIPTOR contains something of the form 'spec,List(0)', then there are no objects matching the spec.

14.2 Messages which Span Physical Message Boundaries

<<<to be supplied>>>

14.3 WM-SHOW-NAMES

The SHOW OBJECTS, SHOW FILES, and SHOW SERVICES commands map into WM-SHOW-NAMES.

```
WM-SHOW-NAMES(id,obj-typ,List(spec1,spec2, ... ,specN))
--> obj-typ,2N-List
    (spec1,List
      (objname-11, ... ,objname-1m),
    spec2,List
      (objname-21, ... , objname-2n), ...
    specN,List
      (objname-N1, ... , objname-Np))
```

where:

spec1, ... specN (CHARSTRs) are the specs given by the user.

Returned:

Objname-ij (CHARSTR) is the j-th object satisfying the i-th spec. If any list in the reply message is null, no objects match the associated spec.

14.4 WM-SHOW-DESCRIPTOR

The SHOW DESCRIPTOR command maps into WM-SHOW-DESCRIPTOR.

```

WM-SHOW-DESCRIPTOR(id,obj-typ,List(spec1, ... ,specN),
    fld-sel,List(fldname1, ... ,fldnameM))
--> obj-typ,2N-List
    (spec1,2K-List
        (objname1,List
            (fldname111,qdata111,values111, ...
            fldname11m,qdata11m,values11m),
        objname2,List
            (fldname121,qdata121,values121, ...
            fldname12m,qdata12m,values12m), ...
        objnameK,List
            (fldname1K1,qdata1K1,values1K1, ...
            fldname1Km,qdata1Km,values1Km))
    (spec2,2L-List
        (objname1,List
            (fldname211,qdata211,values211, ...
            fldname21m,qdata21m,values21m),
        objname2,List
            (fldname221,qdata221,values221, ...
            fldname22m,qdata22m,values22m), ...
        objnameL,List
            (fldname2L1,qdata2L1,values2L1, ...
            fldname2Lm,qdata2Lm,values2Lm)), ...
    (specN,2P-List
        (objname1,List
            (fldnameN11,qdataN11,valuesN11, ...
            fldnameN1m,qdataN1m,valuesN1m),
        objname2,List
            (fldnameN21,qdataN21,valuesN21, ...
            fldnameN2m,qdataN2m,valuesN2m), ...
        objnameP,List
            (fldnameNP1,qdataNP1,valuesNP1, ...
            fldnameNPM,qdataNPM,valuesNPM)))

```

where:

spec1, ... specN (CHARSTRs) are the specs given by the user; fld-sel is obtained from the subcommand (default = ALL); the list of field names are those to be included or excluded and is obtained from the user, or if fld-sel = 0, the list is null.

Returned:

The objname fields are the object names to which each spec resolves to. The 'valuesij' fields have arbitrary complexity; if they contain embedded structure, the preceding 'qdataij' will be False and all information should be represented by embedded

fldname-qdata-value triples. The FE will test NSW8 element types in parsing through 'valuesij' fields. If a 'qdataij' is False, then 'fldnameij' is the name of a data structure, e.g., file descriptor (in this case, the elements of the data structure are the elements of a file descriptor, represented as fldname-qdata-value triples). If 'qdataij' is True, then 'valuesij' is a single parameter or List of parameter; the FE must test to see which.

If fld-sel is 'except', the WM will return all unprivileged information in the entry except those specified in the field list sent to it by the FE.

14.5 WM-SHOW-PUBLIC

The SHOW PUBLIC and SHOW SYSTEM commands map into WM-SHOW-PUBLIC.

```
WM-SHOW-PUBLIC(id,List(fldname[,List(arg1, ... ,argN)])
--> List(fldname,qdata,values)
```

here:

fldname is (CHARSTR) either 'scope', 'permissions', 'hosts', 'services', or 'users'. If fldname is 'users', then there are two following arguments in the List: project-name (CHARSTR) and node-name (CHARSTR). The WM will recognize * in either of these arguments as meaning 'all'. If fldname is 'services', then the <service-spec>s in the user command will become elements of the List following fldname. A list of zero elements is treated as referring to all services in the public region; for fldname 'services', a following List is mandatory. The other possible fldnames take no arguments.

Returned:

A fldname-qdata-values triple; see above under WM-SHOW-DESCRIPTOR.

14.6 WM-SHOW-NODE

The SHOW NODE command maps into WM-SHOW-NODE.

```
WM-SHOW-NODE(id,node-spec,authentication,fld-sel,  
             List(fldname1, ... ,fldnameN))  
--> List(fldname1,qdata1,values1, ... ,  
         fldnameP,qdataP,valuesP)
```

where:

fld-sel is obtained from the subcommand (or default = ALL), and fldname1, ... fldnameN are the fields to be included or excluded (null, if ALL fields requested) and is obtained from the user. Currently, node-spec is constrained to be the current user's logged in node and is represented by a CHARSTR of length 0.

Returned:

The fldname-qdata-values triples are as described above for WM-SHOW-DESCRIPTOR.

14.7 WM-SHOW-SESSION

The SHOW SESSION, SHOW SCOPE, and SHOW PERMISSIONS commands map into WM-SHOW-SESSION.

```
WM-SHOW-SESSION(id,sess-index,fld-sel,  
                List(fldname1, ... ,fldnameN))  
--> List(fldname1,qdata1,values1, ... ,  
        fldnameP,qdataP,valuesP)
```

where:

fld-sel is obtained from the subcommand (or default = ALL), and fldname1, ... fldnameN are the fields to be included or excluded (null, if ALL fields requested) and is obtained from the user. Currently, sess-index is constrained to be the currently logged in user session and is represented by a Index of value 0.

Returned:

The fldname-qdata-values triples are as described above for WM-SHOW-DESCRIPTOR.

14.8 WM-MODIFY-DESCRIPTOR

The SET DESCRIPTOR and MODIFY DESCRIPTOR commands map into WM-MODIFY1-DESCRIPTOR and WM-MODIFY2-DESCRIPTOR.

```
[for the ADD and REMOVE operations]
WM-MODIFY1-DESCRIPTOR(id,obj-typ,obj-spec,fld-op,fld-nam,
    List(value1, ...valueN))
--> dstring
```

```
[for the SET and REPLACE operations]
WM-MODIFY2-DESCRIPTOR(id,obj-typ,obj-spec,N-List
    (fld-nam1,old-list1,new-list1),
    (fld-nam2,old-list2,new-list2), ...
    (fld-namN,old-listN,new-listN))
--> dstring
```

where:

[for MODIFY1] fld-nam is the field name the user is modifying and the valuei list is the list of values to be added or removed from the field name in the WM data base entry.

[for MODIFY2] a Replace fld-op is understood. For each field name gathered from the user's command, the old list (currently in the WM data base) is replaced with the new list. If old-list is a list of zero elements, the WM will replace the current contents of the WM data base entry with new-list; in this case, N is always 1.

Returned:

A string indicating the disposition of the request.

14.9 WM-MODIFY-NODE

The SET NODE and MODIFY NODE commands map into WM-MODIFY1-NODE and WM-MODIFY2-NODE.

```
[for the ADD and REMOVE operations]
WM-MODIFY1-NODE(id,fld-op,fld-nam,
                List(value1, ...valueN))
    --> dstring

[for the SET and REPLACE operations]
WM-MODIFY2-NODE(id,N-List
                (fld-nam1,old-list1,new-list1),
                (fld-nam2,old-list2,new-list2), ...
                (fld-namN,old-listN,new-listN))
    --> dstring
```

where:

the same conventions as used in WM-MODIFY?-DESCRIPTOR above are observed.

Returned:

A string indicating the disposition of the request.

14.10 WM-MODIFY-SESSION

The SET SESSION, MODIFY SESSION, and MODIFY SCOPE commands map into WM-MODIFY1-SESSION and WM-MODIFY2-SESSION.

```
[for the ADD and REMOVE operations]
WM-MODIFY1-SESSION(id,fld-op,fld-nam,
                  List(value1, ...valueN))
    --> dstring

[for the SET and REPLACE operations]
WM-MODIFY2-SESSION(id,N-List
                  (fld-nam1,old-list1,new-list1),
                  (fld-nam2,old-list2,new-list2), ...
                  (fld-namN,old-listN,new-listN))
    --> dstring
```

where:

the same conventions as used in WM-MODIFY?-DESCRIPTOR above are observed.

Returned:

A string indicating the disposition of the request.

14.11 WM-DELETE

The DELETE command maps into WM-DELETE.

```
WM-DELETE(id,obj-typ,object-spec,m-attr-list,hlpdsp)
--> dstring
```

where:

obj-spec is the spec of the object to be deleted and is obtained from the user. If the spec contains a "/type=<type>" string, it is stripped off by the FE and (in the case of OBJECTS) used to determine obj-typ. Other attribute strings in the spec are retained. The FE will reject any inconsistent attribute specifications without sending them to the WM.

Returned:

A string indicating the disposition of the request.

14.12 WM-COPY

The COPY command maps into WM-COPY.

```
WM-COPY(id,obj-typ,old-obj-spec,m-attr-list,
        new-obj-spec,s-attr-list,hlpdsp)
--> dstring
```

where:

As with DELETE, neither old-obj-spec nor new-obj-spec may contain a type attribute specifier. If either spec as given by the user contained a type attribute specifier, it is stripped off and (in the case of OBJECT) used to determine obj-typ.

Returned:

A string indicating the disposition of the request.

14.13 WM-RENAME

The RENAME command maps into WM-RENAME.

```
WM-RENAME(id,obj-typ,old-obj-spec,m-attr-list,  
          new-obj-spec,s-attr-list,hlpdsp)  
--> dstring
```

where:

old-obj-spec and new-obj-spec do contain a type attribute specifier (see above in DELETE and COPY)

Returned:

A string indicating the disposition of the request.

14.14 WM-PLACE-IMAGE

The PLACE command maps into WM-PLACE-IMAGE.

```
WM-PLACE-IMAGE(id,hlpdsp,disambig-control,  
               confirm-control,object-spec,m-attr-list,  
               lkup-hint,dest-site,qretain)  
--> dstring[,List(site1,site2, ... siteQ)]
```

where:

disambig-control and confirm-control are NSWB8 Booleans governing the presence or absence of disambiguation and confirmation dialog, respectively (see Sections 4.2.2.34 and 4.2.2.35 in the WM Subsystem Specification document); dest-site is a NSWB8 Integer (32-bit) giving the long-form ARPANET host number of the destination site; qretain is a NSWB8 Boolean which, if True, specifies that the image at the origin host be retained.

Returned:

A string indicating the disposition of the request; if the request succeeded, a List of sites at which images of the reside is appended.

14.15 Starting Up a NSW Service

The USE command maps into WM-RUNSERVICE; in the service start-up protocol scenario, FM-BEGINSERVICE or FM-REBEGINSERVICE are also involved.

```
WM-RUNSERVICE(id,<user name for service session>,2N-List
    (service-spec1,
        List(param-11, ... ,param-1n),
    service-spec2,
        List(param-21, ... ,param-2m), ...
    service-specN,
        List(param-N1, ... ,param-Np)))
--> dstring,service-addr,svid,
    List( static tool params )
```

where:

service-spec1, ... service-specN (CHARSTRs) are the service specs given by the user; param-ij (CHARSTR) is the j-th parameter to be passed to the i-th service; it is also obtained from the user. If a service is invoked with a null parameter list, its corresponding parameter list in the message is a list of zero elements.

Returned:

dstring is a character string indicating the disposition of the request; service-addr is the MSG process name of the Foreman/Service; svid is a NSWB8 Integer assigned by the WM uniquely identifying the service instance; and 'static tool params' is control information concerning the FE-TBH interface as specified in the catalog entry for the service (if any).

```
FM-BEGINSERVICE(id,FE-addr,cr-addr,user-id,  
                 user-name,N-List(services))  
--> qstart,List(wsd,access-info),service-addr
```

where:

FE-addr is the MSG process name of the associated FE;
cr-addr is <<<to be supplied>>>; user-id is <<<to be supplied>>>;
user-name is <<<to be supplied>>>; services is
List(service-descriptor,entvec,list(dynamic parms)) and
service-descriptor is (service-name,service-scope,host,prog-name,
service-type, list(static parameters)); host is a NSWB8 Integer
giving long-form ARPANET host number of service-bearing host.

Returned:

qstart is a NSWB8 Boolean specifying whether the requested
service has started or not; wsd is name (host-dependent) of
work-space directory assigned to service; access-info is
host-dependent information relating to accessing the service and
workspace; service-addr is MSG process name of the
Foreman/service invoked.

```
FM-REBEGINSERVICE( <same as FM-BEGINSERVICE> )
```

14.16 WM-LOOKUP-FILE

The TYPE, PRINT, and GET commands map into WM-LOOKUP-FILE;
the protocol scenario (see Section 18.2.1) involves other
procedure calls also.

15 TBH/WM DATA BASE SYNCHRONIZATION

Currently, when a TBH restarts, the reliability scenarios call for it to report to the WM only new LND saved sessions (i.e. those not yet confirmed by the WM) which are rerunnable, and those recent sessions which the FM is discarding because they are not rerunnable. As currently formulated, the WM-LND-MAINT primitive will accept either a single tool-id or a list of tool-ids. In either case, the list is assumed to be incremental i.e. is in addition to whatever saved sessions are already in the data bases. This has led to situations where data base entries stored in one data base (e.g. WM) but not in the other (e.g. FM) required manual periodic deletion since they were not useable and were unreportable as deleted. In addition, hosts which did not save any sessions across TBH restarts had no simple way to automatically clean up invalid WM data base entries.

The modification we propose for NSW 6.0 is to add a saved session synchronization form of the WM-LND-MAINT call to allow a TBH to report ALL of the saved sessions it knows about (or is willing to save) at once, including those previously confirmed as rerunnable by the WM. This would allow the WM to purge its tables of all entries which the hosts have not indicated a willingness to save and are thus effectively useless. On completion of this transaction, the WM and TBH service session data bases should be synchronized. In particular, this form of the LND synchronization transaction will be required at TBH restart time. A host that does not support rerunnable sessions will report no saved sessions, and the WM will purge all current service activation records for that host.

There will now be two WM functions to support workspace reliability scenarios, WM-LND-MAINT and WM-LND-SYNCH. The WM-LND-MAINT message provides only for an incremental update to the WM's tables. Both the single-session and multiple-session variants of WM-LND-MAINT for continued use by an FM implementation. A new protocol message (WM-LND-SYNCH) is provided for complete resynchronization of the FM and WM tool session data bases. The intent is that the WM-LND-SYNCH be used at a minimum whenever the TBH restarts, and that the WM-LND-MAINT may be used during normal operation of the system to report isolated (e.g. single service session) updates without requiring a full resolution of the TBH database.

WM-LND-MAINT:

```
list: (string:HERALD
      list: (integer:TOOLID...))
```



```
list: (integer:TOOLID...)
)
```

After the FM herald, the first list is of TOOLID's of tool sessions to preserve, the second list is of tool session to discard. Either list may be null. The most typical case called the single-session case, occurs when only one list is non-null, and contains only a single TOOLID. It is typically used to handle failures relating to a single tool instance (e.g. a broken FE connection) without incurring the overhead of a complete TBH data base scan.

<reply:

```
list: (integer:TOOLID...)
```

The reply is a list of "exceptions." These sessions are unknown to the WM or have been previously discarded. Their entries should be discarded by the FM. Should an entry appear on both the "preserve" and "discard" lists of the WM-LND-MAINT message, the WM shall discard the session.

WM-LND-SYNC:

```
list: (string:HERALD
      list: (integer:TOOLID...))
)
```

The list of TOOLID's specifies the entire set of rerunnable tool sessions known to this TBH FM. By implication all other sessions associated with this TBH should be discarded. Up to about 500 sessions may be specified in a single MSG message under current size limitations. This primitive must be invoked by each TBH on every TBH restart.

<reply>:

```
list: (integer:TOOLID...)
```

The reply is a list of exceptions which the WM instructs the TBH to discard.

16 AUTOMATIC CLEANUP OF EXISTING LOGIN SESSION RECORD ON RE-LOGIN ATTEMPT

This user interface and protocol modification is designed to support the ability to have a user request that an old inactive session activation record in the WM data base, indicating a session already in progress, be purged so that a new login request can be processed. In the current system, such an old, inaccurate session record prevents a user from re-logging into the system. This protocol change would also support the transfer of the service context from the old session to the new one within the limitations posed by the LND saving mechanism already implemented in the current system.

User-Interface changes:

Instead of seeing an FE error-printout of the WM's text "Somebody is logged in to Node <proj>+<node>", the user will see a message more like:

- o NSWExec records indicate that <proj>+<node> is occupied with a session which started at <timestamp>. Type CR to save that session and continue your login, or ^X to abort the login.

Assuming the user types CR, his login will complete as usual; if there were in fact running Foremen with LNDs to save from the previous session, the user will see -- probably after some minutes -- messages announcing the RESUMEability of those saved sessions ("Records are available...").

Front-End changes:

The WM will send an Alarm to the FE, with a code indicating that it should do a Stopme, without sending any other messages. or waiting for any replies. If the FE chooses not to accept, or respond to, this alarm, it will become disconnected from all its pending tools, and become unknown to the WM (its Sessid (user id) will no longer retrieve a valid Active User Entry).

Data-Base changes:

- + Include an NSW-timestamp element in Node records and Active User entries. This allows the printout mentioned above, which helps the user recall whether the wedged session is actually one he had trouble with, or is someone else actively logged into the node. It also permits the normal LOGIN reply to give the time of last login, which is a small, but interesting security datum.

Protocol changes:

- + New FE Alarm command, mentioned above.
- + Foremen must be able to accept an FM-SAVE-LND call from Works Managers, as well as from FEs, and send their reply back to the sender.
 - Foremen need not change their present practice of sending Generic WM-LND-`SAVED` messages during the `SaveLnd` scenario. The fact that some other instance of the WM will receive and process those messages is logically irrelevant, and will in fact simplify the modifications of the WM code.

Works Manager changes:

- + At the outermost block level, declare a flag to indicate peremptory overlogging, and space to contain the necessary information from the apparently-busy Node record and from the corresponding AUE, if available: At least Sessid, FE Process name, Semaphore list.
- + After the LOGIN parameters have been validated, and the user's Node record has been retrieved, the Sessid field (`NODE>>node.ndsessid`) is tested.

The present code makes an error exit; it should be changed to set the overlogging flag, record the Sessid, and try to retrieve the AUE for that Sessid. If it can find the AUE, it should copy the essential information (or make a local copy of the whole AUE) and delete the old AUE from the Data base. Then it should return to the present code, setting up a new AUE.
- + Before writing the new AUE into the database, test the overlogging flag and, if set, append to the list of files with semaphores set (which has just been copied from the Node record into the new AUE) the names of any such files which were in the wedged AUE and are not already on the list.
- + Return to the present code through sending the `WM->FE:Reply` (to the `FE->WM:WM-LOGIN` call). Then, before sending the `WM->FE:FE-LND-SAVED` calls for the old tools found in the Node record, test the overlogging flag, and if it is

set, generate a list of all ToolIDs which have the saved (wedged) Sessid as their user's id.

Then, for each element of that list:

- Fetch the Active Tool Entry for that ToolID to verify that it is (still) "active" and still points to the Sessid in question;
 - Construct a WM->FM:FM-SAVE-LND call to the Foreman process named in the ATE; close-and-unlock the ATE (so that the WM-LND-MAINT code can get at it); send the message, and wait for the reply.
 - Repeat the above steps for the next ToolID on the list, if any.
- + Then continue the remainder of the present code in Wmlog, sending WM->FE:FE-LND-MAINT messages for the older tools which had been in the Node record when it was originally fetched. (The normal processing of the FM->WM:WM-LND-MAINT messages will have caused WM->FE:FE-LND-MAINT messages for these newly-saved tools to be sent, already -- presumably these are of greatest interest to the user.)
- + Send the above-mentioned MSG Alarm to the FE Process named in the wedged AUE.

17 WORKSPACE COMMAND INTERPRETERS and GENERALIZED SERVICE SUPPORT

NSW 6.0 will support Workspace Command Interpreters (WS-CI) to allow more direct user control over NSW service sessions, and as support for providing native mode tool execution, tool kits, and detachable service sessions. When a WS-CI is provided by a TBH, it is activated in one of three ways:

- o Through a designated well known name for the WS-CI entered in the NSW object catalog, and activated with the USE command.
- o Through a designated NSW-wide control sequence issued while some other service is active.
- o Through the invocation of a tool kit (collection of tools to be run on a single host); whenever a tool kit

is invoked, the user will find himself interacting with a WS-CI to coordinate the individual tool activations.

A WS-CI will use as its prompt the host name on which it runs. All WS-CI's will provide the following commands:

1. GET (NSW-file) as (Workspace file)
2. DELIVER (workspace file) as (NSW file)
3. RUN (program name)
4. RESUME (program name)
5. KILL (program name)
6. QUIT

In addition, WS-CI's may also support standard commands of the following form:

7. SHOW WORKSPACE-files ; to view the names of workspace files
8. SHOW TOOLS ; to view the names of tool kit members
9. SET file-access-mode
10. SAVE workspace-session ;to save the workspace context for later resumption

There may also be any number of host specific WS-CI commands, as well as interfaces to other NSW functions supported by the central NSW system.

Additional parameters which are to be added to the appropriate session setup messages to support extended services and more sophisticated user interfaces to services:

- o An optional user-specified list of character strings collected by the FE and passed to the FM/SERVICE on startup (i.e. system can collect service parameters on command line; useful for non-TOPS-20 tools);
- o Full service name (character string); scope (region)

name was found in (character string); user login name (character string) ; user session id (integer); NSW catalog name for workspace (character string when supported);

All passed to FM as part of SERVICE startup to support the development of more sophisticated tools and tool interfaces;

- o Begintool handling a list of augmented tool descriptors to be processed by a WS-CI as possible arguments to a local RUN command;
- o Augment WM to FE Begintool response to support arbitrary cataloged ICP tools; the FE will do ICP to the host/socket returned from the Begintool.
- o Host characteristics passed to FE on service session startup; (static through descriptor and/or dynamic through communication setup); Augmented FE-Open Conn to support dynamic setting of host characteristics regarding communication with the particular host/service e.g. user interface for Telnet control options, an alarm version of WS-CI "^C" if necessary, etc.
- o Augmented tool type parameter values

17.1 User Interface Changes for Service Control

The following NSWExec commands will be modified/added to support service invocation and control.

- o USE <session-name>=<list of service args>
- o SAVE <session-name>
- o TERMINATE <session-name>
- o ABORT <session-name>

17.2 CHAINING TOOLS IN EXISTING WORKSPACE

We are adding an additional WM lookup function to support a form of workspace tool chaining. The Access-to-Service-Object primitive would be a call on the WM to perform lookup and access control for an NSW service, in much the same way that

Access-to-File-Object looked up NSW catalog files.
Access-to-Service-Object would return a service descriptor, much like Access-to-File-Object returns a file descriptor. A Workspace command interpreter could use an Access-to-Service-Object primitive to support a WS-CI command for running a second (or subsequent) service in an existing workspace. For NSW 6.0 tool chaining will be available only for co-located services on hosts supporting a WS-CI.

ACCESS-TO-SERVICE-OBJECT

This function ACCEPTS as arguments:

Identification

- o Session id; service id; or {user + password};

Exception Handling

- o Help process;
- o Disambiguation control
- o Confirmation control

Logical

- o Service spec (subject to general lookup rules);
- o Attribute list (implicit type = service; site = expected to be same as requestor);
- o Access type (execution);
- o Lookup hint

This functions RETURNS as results:

Confirmation

- o Success/failure

Logical

- o Full NSW service name of object looked up;
- o Scope used (if any) to locate the object;
- o Lookup hint

Physical

- o Physical service descriptor from catalog (similar to what gets transferred with BEGIN TOOL TRANSACTION).

NOTES:

Service object lookup is performed in exactly the same manner as file object lookup. Once an appropriate object is identified, the accessing agent requires an execute key for the operation to succeed.

17.3 DETACH SERVICE SESSION

This protocol addition would allow a Foreman to request that a service connection be smoothly terminated, as if the session had ended but without the tool termination interaction with the WM which usually accompanies a message ending the conversation. The intent here is to allow a WS-CI on a TBH which provides rerunnable Service Sessions to support a SAVE command. When available, this command instructs the WS-CI to save the workspace context so that it may be resumed (reattached to) at some later time. The current reliability scenarios support all phases of this except for the smooth cessation of the session without invalidating the session record. The FM can use the single tool instance of the LND save function to perpetuate the service session record. However, currently, the FM can not break the connection to the FE without invoking error recovery procedures (if it unilaterally breaks the connection) or invalidating the service entry with the WM (if it sends an Endtool message.)

The new FE function is described next.

17.3.1 FE-CLOSECONN

This procedure call is sent to the FE by a FM in response to

a user-initiated SAVE session command to the workspace command interpreter. On receiving such a request, the FE will close its connection to the service and purge its records of the service activation. The FM is responsible for reporting the saved session to the WM, which may subsequently alert the FE to its rerunnability.

FE-CLOSECONN(svid,qterm,service-addr,conn-id) -->

where:

svid is the WM-assigned service id; qterm is NSWB8 Boolean specifying whether service-addr is the MSG process name of the service to which the connection is being closed; conn-id is the connection id defined at service startup.

The FE will send a null reply message to the FM.

18 FILE-HANDLING CAPABILITIES for the UNIX FE

This section discusses the addition to the UNIX NSW FE of commands for handling NSW file 'TYPE' and 'PRINT' functions. The functions occur in three phases:

1. actually importing the file to UNIX;
2. displaying the file on the user's terminal or sending it to the UNIX printer; and
3. deleting the file.

Handling the commands in this manner guarantees that the user will not receive an incomplete listing or see only part of the file typed on his terminal. It also is more efficient than sending the file directly to the user's terminal or the line printer, as the File Package which is sending the file need not be delayed by the relatively slow output devices. Also, it makes the addition of a 'GET' command to transfer files from NSW file space to UNIX file space trivial, the difference between 'GET' and 'TYPE' or 'PRINT' being that the user specifies the UNIX filename in the case of a 'GET', whereas the FE specifies it for a 'TYPE' or 'PRINT', and a 'GET' completes after step 1 above.

Immediate-return mode

In immediate-return mode, the user will see a "Command initiated" message when he issues the TYPE command and sometime later will see a "Command complete, output waiting" message. At this point, he may issue a DISPLAY command to view the output, or a DISCARD command to discard it. If, while DISPLAYing the output, the user types ^X, the DISPLAY will be aborted, the UNIX copy of the file deleted, and the rest of the output discarded.

18.1.2 PRINT command

The syntax of the PRINT command will (initially, until line printers become a catalogued resource) be the same as the TYPE command:

```
NSW: PRINT (NSW filespec) UMMA.GUMMA
```

Deferred-return mode

In deferred-return mode, the user will issue the PRINT command and receive the standard "Command initiated" prompt from the Front End. The file importation will then take place, and when that has completed, the tool process of the FE will fork a spooler process and pass the file name to it. When the spooler process terminates (completes), the Front End will issue the "Command complete" message and request another input command.

Immediate-return mode

In immediate-return mode, the Front End will indicate that the command has been initiated and then return control to the user. At some later time, it will indicate that the command has completed (which means that the file has been spooled, not necessarily that it has been listed yet).

18.1.3 GET command

A GET command will look something like:

```
NSW: GET (NSW filespec) BONZO (as UNIX file) /usr/fred/ox
```

The User Process of the FE provides the user interface for the new commands. The Protocol Process implements the message communication with other NSW components, and the Tool Process implements the actual NSW file transfer.

18.1 User Interface

This section describes what the user is expected to see when he issues one of the commands TYPE, PRINT (or GET).

In all three new commands (as in existing FE commands), if the command is issued in deferred-return mode and ^X is typed before the command completes, the command will be treated as if it had been issued in immediate-return mode, and the user will be prompted to input the next command.

If ^T is typed in either deferred or immediate return mode, the FE will list all currently active commands, including the TYPE, PRINT or GET, which will be listed as being "In progress."

18.1.1 TYPE command

The format of the TYPE command will be something like:

NSW: TYPE (NSW filespec) MOO...Z...OOM

Deferred-return mode

When issued in deferred-return mode, the FE will issue the standard "Command initiated" prompt and begin the file import scenario described below. At this point, he may see help messages asking him to disambiguate the file spec or indicating that the requested file is not a text file. Otherwise, the user will have to wait until the file transfer has completed, whereupon the FE will announce "Command complete" and start to type out the contents of the file.

If ^X is typed while the file is actually being typed out, (note that this is after the command has completed, and the FE is automatically DISPLAYing the output of the command), the TYPE command will be aborted, the UNIX file deleted, and the rest of the output discarded. This means that if the user re-issues the TYPE command, the file must be reimported to UNIX.

18.2 Importing the File to UNIX

The NSW protocol scenario for importing the file to UNIX is identical (or nearly so) for both the 'TYPE' and 'PRINT' functions (and for the 'GET' function). It is:

The FE-FP protocol for file transfer is identical to the current FP-FP subscenario for NSW file transfer.

To select a PCD, the FE will look for a copy of the file on the local host (which it will never find, since there is currently no File Package for UNIX), then will look for a family copy (which it will also never find), then will look for a copy that is already IL-encoded, and if no such file is found, will choose any other copy of the file.

C-98

invalid file encodement), the Front End will abort the operation by closing the connection, deleting the file, and issuing an error message to the user.

The file has, by the preceding scenario, been copied into UNIX file space, having a UNIX file name given by the FE (or in the case of a 'GET' command, by the user). The NSW file name is known to the FE.

18.2.2 Implementation

Implementation is straightforward and involves few (if any) changes to the FE architecture.

Additions to FE Tables

The following tables must have entries added:

- o Grammar Tree (user process)
- o Protocol Scenario Definition Table (protocol process)

Also, a few new symbol definitions must be introduced.

Additions to Existing C Functions

The following existing C functions must have new 'case's added:

- o MakRmMs (user process) - assembles initial WM NSWTP message
- o AssemMs (protocol process) - assembles FP-SENDME message
- o In all three FE processes, functions which read from pipes must have the 'get file' pipe message capability added.

The following existing C functions must be slightly altered:

- o ToolInt (tool process) - add capability to write to file

- o IActivM, OpnTool, et. al (tool process) - add capability to handle file transfer capability
- o In all three processes, there may need to be scattered alterations to deal with the new type of conversational partner being defined (the "file" partner)

New C Functions

The following new C functions must be written:

- o (protocol process) - process reply to initial WM call; involves selecting proper PCD from list
- o (protocol process) - process reply to FP-SENDME
- o (tool process) - IL to UNIX translator

18.3 Viewing/Listing the File

18.3.1 Design Aspect

The UNIX file name of the imported NSW file will be generated by the User Process (from its process id and the command number - this guarantees a unique name even if several FEs run out of the same directory), or (for GET) specified by the user to the User Process, and communicated to the Tool Process as a following string to the 'get file' pipe message.

The User Process will keep a pointer to the NSW filename in the User Session Table.

In the case of the PRINT command, the Tool Process will, upon completion of the file transfer, fork a process which will spool the file to the UNIX printer.

18.3.2 Implementation

The DISPLAY command must be modified to display the file on the user's terminal. This is a fairly straightforward modification, requiring only that the User Process reconstruct the UNIX file name of the imported NSW file in case the output of a TYPE command is being displayed.

19 MISCELLANEOUS OTHER CHANGES

- o MSG extended host addressing modification
- o Fix the password leaking with recorded message traffic
- o TOPS-20 Foreman using global file types
- o UCLA Batch system enhancements
- o Directly runnable TOPS-20 FE

NSW SYSTEM PERFORMANCE EVALUATION

BBN NSW Note No. 29

Richard E. Schantz

February 1980

Appendix D

NSW System Performance Evaluation

1. Summary

BBN has recently completed a series of experiments toward evaluating the performance of the current NSW system (version 4.0) under various host configurations. This work was undertaken partly to help us better understand the relationships between configuration parameters and system performance measures, and partly to evaluate system models developed by the U/Texas performance group.

In summary, these experiments confirm our expectations: that one can achieve excellent NSW response characteristics with a large enough single host configuration and a limited load. However, the extensiveness of the computer resource demands by NSW processes in carrying out NSW operations, the causes of which are many, severely limit the achievable throughput, and make such configurations extremely non-cost-effective. The current NSW implementation suffers in a cost-benefit analysis due to the lack of a large inventory of useful, distributed resources which might temporarily compensate for the large overhead associated with NSW operations. The local resource special case is, and is likely to continue to be the most heavily exercised path, leaving the easily compared native host alternative to emphasize the NSW/non-NSW performance disparity.

In Sections 2 thru 5 of this note we report on a selected set of controlled experiments and their results. Section 6 is a discussion of the results, while Section 7 provides performance data from a random sampling of the operational user system under uncontrolled conditions. Sections 8 and 9 take a closer look at typical NSW component resource demand patterns to understand why the demand is so extensive. Finally, Section 10 explains an often unnoticed part of the current implementation which may prove to be a key element in improving the overall performance of the system.

The reader is assumed familiar with the NSW concept and its current system architecture, including the system organization of its major support host operating system, TOPS-20. For a detailed discussion of these topics, see "Operating Systems for Computer Networks," IEEE COMPUTER, January 1978, "The NSW: A Distributed Processing System," ACM National Conference Proceedings, fall 1977, and "A Performance Analysis of the NSW System," BBN Report No. 3847, March 1979.

AD-A185 967

NSW (NATIONAL SOFTWARE WORKS) PERFORMANCE ENHANCEMENTS
(U) BOLI BERANEK AND NEWMAN INC CAMBRIDGE MA
R E SCHANIZ ET AL. JAN 81 BBN-4600

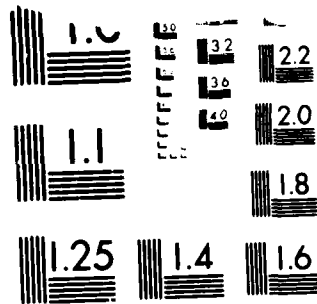
4/4

UNCLASSIFIED

F/G 12/3

NL

END
DATE
FILMED
8



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

2. The Experiments

For each of three different physical host configurations of the TOPS-20 computer system, a self-contained NSW system was configured. In addition to the NSW processes and required TOPS-20 "system" jobs, there were three background measurement programs which periodically sampled aspects of the TOPS-20 System performance. One measurement program sampled overall system performance measures, another program sampled the memory utilization to determine the effects of memory sharing, and the third program represented a fixed periodic background task for measurement purposes. There was no system load beyond NSW "users" and the measurement programs. Our experiments consisted of running one, then two, then three simultaneous user scripts against each of the NSW configurations. A user script consisted of NSW login, running a tool (TECO), six consecutive NSW OPEN scenarios (unambiguous filespec, small file), six consecutive NSW DELIVER scenarios (unambiguous entryname, small file), tool termination, and NSW logout. The user scripts were contained in files on a single non-NSW host, and drove the NSW Front End via remote Telnet connections. When simultaneous NSW users were simulated, the script drivers coordinated to issue their "USE TOOL" request at the same wall clock time. After this initial coordination, the NSW scripts were not further resynchronized.

3. The System Configurations

Name	RADC-old(2040)	RADC-new(2040)	BBND(1090T)
Processor	KL	KL	KL
Cache	no*	no*	yes*
Memory	MB20*	MH10*	variable*
Nominal Memory	256K words	512K words	1000K words
Memory Swapping			
Pages	324	830	1853
Swapping Disks	2	2	8
Channels	1	1	1

Table 1: The hardware configurations for the host systems

* MH10 memory is slower than MB20 memory, which can only be used with an internal bus structure. The BBND system has a mixture of various speed memories on an external bus structure. The slower memory, for our purposes, translates into a slower CPU.. Likewise, the cache utilizing fast memory components, translates into a faster effective CPU. It is difficult to assess exactly the different configurations in this regard because of the variability of reads and writes, and because of the probabilistic effects of cache hits. Through a combination of sample code segment timing comparisons, and advertised speeds of the various memory devices, we have approximated the CPU differential as follows:

if RADC-old CPU unit time is represented as 1, then

an RADC-new CPU unit is 1.25, i.e., new is 25% slower than old, and

a BBND CPU unit is .455, i.e., a cache seems to make the CPU approximately 2.2 times faster.

The RADC configurations ran the standard DEC TOPS-20 release 3 operating system. The BBND system ran the TOPS-20 release 101B, with BBN modified processor and memory management routines.

4. Selected Response Time Data

The following table summarizes a comparison of the user response times observed for a single, typical NSW operation (GET FILE), on a dedicated NSW host, with 1, 2, or 3 NSW users simultaneously exercising the same transaction script. The file was catalogued by the WM and resident in the local host segment of the NSW file space. The timed interval is the period beginning when the NSW (the Foreman) has collected the candidate file name, and ending when the file is made available to the tool. All times are in seconds. There were six trials per user.

<u>RADC old</u>	<u>1 user</u>	<u>2 users</u>		<u>3 users</u>		
minimum time per user:	U1=13	U1=30	U2=32	U1=34	U2=37	U3=37
maximum time per user:	U1=22	U1=43	U2=41	U1=61	U2=55	U3=60
average time per user:	U1=17	U1=36	U2=36	U1=46	U2=45	U3=47
 <u>RADC-new</u>						
minimum time per user:	U1=9	U1=13	U2=11	U1=22	U2=22	U3=24
maximum time per user:	U1=13	U1=24	U2=23	U1=45	U2=41	U3=39
average time per user:	U1=10	U1=18	U2=16	U1=35	U2=32	U3=30
 <u>BBND</u>						
minimum time per user:	U1=7	U1=8	U2=8	U1=11	U2=11	U3= not
maximum time per user:	U1=9	U1=11	U2=12	U1=16	U2=17	available
average time per user:	U1=7	U1=10	U2=10	U1=13	U2=13	

Table 2: Response time for the NSW GET FILE operation under various host configurations and NSW load conditions.

5. Overall System Utilization for Repeated Scenarios

The following table reports samples of overall system utilization during peak periods of NSW activity for each of the configurations during consecutively repeated NSW operations by 1, 2, and 3 users. Most measurements are one minute samples (i.e., represent the preceding minute interval of activity) during which there was continuous and consecutive NSW transaction invocation.

<u>RADC-old</u>	<u>1 user</u>	<u>2 users</u>	<u>3 users</u>
% of interval in scheduler	11%	18%	23%
% of interval sold as user CPU time	37%	45%	51%
% of interval CPU busy = (sched. + sold)	48%	63%	74%
% of interval in swap wait (CPU idle)	52%	37%	26%
total secondary memory traffic (pages/second)	46	66*	66*
total swapping requests (#pages)	1458	2174	2088
mean real time to process a page fault (in milliseconds)	30	61	66
throughput (approx. # of NSW operations completed per minute)	3	4	4
response time of background task-no NSW load (in seconds)	.15	.15	.15
response time of background task-during NSW load (in seconds)	1.03	2.78	3.14

Table 3a: RADC-old system utilization

*this value represents approximately 100% of system capacity

<u>RADC-new</u>	<u>1 user</u>	<u>2 users</u>	<u>3 users</u>
% of interval in scheduler	15%	20%	25%
% of interval sold as user CPU time	69%	76%	72%
% of interval CPU busy = (sched. + sold)	84%	96%	97%
% of interval in swap wait (CPU idle)	16%	4%	2%
total secondary memory traffic (pages/second)	17	19	33
total swapping requests (#pages)	26	37	606
mean real time to process a page fault (in milliseconds)	40	57	58
throughput (approx. # of NSW operations completed per minute)	6	7	6
response time of background task-no NSW load (in seconds)	.22	.22	.22
response time of background task-during NSW load (in seconds)	.54	.53	1.03

Table 3b: RADC-new system utilization

<u>BBND*</u>	<u>1 user</u>	<u>2 users</u>	<u>3 users</u>
% of interval in scheduler	4%	6%	7%
% of interval sold as user CPU time	36%	43%	48%
% of interval CPU busy = (sched. + sold)	40%	49%	55%
% of interval in swap wait (CPU idle)	58%	50%	44%
# swap requests (pages/second)	0	0	0
throughput (approx. # of NSW operations completed per minute)	9	12	14

* Some of the measurement procedures and values for this system are related to but not necessarily exactly the same as for the RADC configuration due to differences in the operating systems and the programs used to collect the data.

Table 3c: BBND system utilization

6. Discussion

The effects of increased main memory capacity for each of the host configurations can be clearly seen in the response time and utilization figures of the preceding sections. For the small configuration (RADC-old), even for a single user the NSW paging demand outstrips available main memory, causing extensive swap wait delay. For the large configuration (BBND), response time delays are certainly acceptable because of negligible swapping, and a faster effective CPU (due to the cache). However, there still are extensive process and scheduling CPU demands by the NSW processes despite the adequate stand-alone response time.

The response time and utilization measures recorded are slightly worse than the best possible observable NSW performance under the reported configurations because of the concurrent measurement activity and some file manipulation strategies employed in recording the data. Without the measurement activity, we would expect response times to be decreased by a few seconds. For the small configuration, the major effect would be less contention for memory. For the large configurations, the major effects would be less CPU usage and contention, as well as a few seconds less of swap-wait time due to file consistency and page fault delays forced by the data collection strategy (i.e., not by memory availability).

With this in mind, a single user for the medium configuration, and up to three simultaneous users on the large configuration would see what we regard as adequate responsiveness in an absolute sense (i.e., 10 seconds or less per major NSW transaction). However, the cost of the observed responsiveness (table 2) in terms of system resources consumed to achieve it, is quite high. This is shown by the utilization and throughput measures of table 3.

To analyze this data within a simple model, let's assume that NSW transaction processing has resource priority over all other processing (i.e., NSW tools, background use). The no load trials reported in tables 1 and 2 closely approximate a special case of this situation since there is little or no external loading. Taking the small host configuration as an example [RADC-old], we observed a single user transaction throughput of approximately 3 scenarios per minute (i.e., during a period of consecutive NSW transactions, three would complete in a minute of real time). This translates roughly into 180 non-concurrent consecutive transactions per hour. If we conservatively postulate a typical tool session as referencing a single NSW file, and producing a single new NSW file at tool termination, then a tool session has four major NSW transactions associated with it (RUN, OPEN, DELIVER and TERMINATE). [Note that this also models sessions incorporating proposed optimizations such as a multiple tool session (tool chaining), and batched file

requests (e.g., a multiple file delivery transaction)]. If we further postulate a ten minute tool session time, then as few as seven concurrent logged in NSW users running tools in the pattern described above might cause one NSW transaction to be ongoing at all times during an hour's interval (24 NSW transactions per hour per user x 7 users, versus a throughput of 180 transactions per hour).

Under non-concurrent transaction assumptions, each NSW transaction would have a response time of about 17 seconds (from table 2). This response time/throughput would be achieved by consuming 50% of the CPU resource and 75% of the swapping capacity of the system configuration (from table 3a). The result would be that those tool sessions which were not currently engaged in a major NSW transaction (i.e., the 6 others) as well as other non-NSW tasks would be faced with having their system demands met by effectively 1/2 of a CPU and 1/4 of the swapping capacity of the system. This is exemplified in a simple way by the background task response time measurement of table 3a, showing considerably diminished responsiveness. In reality, actual tool load would likely make all of the response times worse than indicated because of increased swapping due to additional memory contention. Similar analyses can be done for the larger configurations to estimate supportable users or likelihood of an ongoing NSW transaction.

As the real time requirement of the NSW transactions approaches 100% of clock time for an observation interval due to increasing numbers of NSW users, the probability of several simultaneous active transactions increases. The multiple concurrent user performance measures of Table 2 would hold for these transactions. That is, if two transactions execute simultaneously, we would expect the response time for each to enlarge to 36 seconds, at a cost of 63% of the available CPU and close to total capacity of the swapping devices during this interval. While these transactions were proceeding, background or tool users would see worse responsiveness than with a single outstanding transaction. For the small host configuration, there appears to be no benefit derived from handling simultaneous transactions. In the postulated workload situation, the expanded response times negate the possibility of extended periods with no active NSW transactions to compensate for simultaneous user activity. The performance of the RADC-old configuration without significant external load, as reported in our experiments, is roughly equivalent to an NSW system with a few hundred pages of memory dedicated to NSW (private memory) for its processing demands.

In the absence of high priority assigned to NSW transaction processing as simulated by no external load, we trade off increased NSW transaction response time against a more equitable distribution

of available system resources to the concurrent background demands. Pie-slicing or class scheduling represents one approach to organized distribution of system resources among different NSW users (although this is not currently done), as well as among NSW and non-NSW users (this is currently done on some NSW hosts).

For the current series of experiments, the large host configurations are roughly equivalent to NSW systems with large amounts of private NSW memory. For these configurations paging is not a major factor for up to two simultaneous users on the medium sized memory system, and for three simultaneous users on the large configuration.

When paging is not a large factor due to a combination of large available (pre-initialized) memory and memory sharing by the NSW processes, there is an advantage to simultaneous NSW transaction requests. The advantages can be seen from the perspective of those processes not involved in NSW transactions and from an NSW transaction throughput point of view. For a small (relatively) additional cost in average response time and slightly decreased short-term CPU availability for non-NSW processing (i.e., an increase in measured CPU utilization for NSW transaction processing), higher throughput rates mean additional time when no NSW transactions are active, given a constant NSW workload. At these times, overall system response and utilization would be

equivalent to the non-NSW host case. However, if NSW processing demands receive high priority to achieve an adequate NSW responsiveness, the extensiveness of the CPU demand (40% over a number of seconds) during an NSW transaction would likely cause a perceptible delay for non-NSW transaction processing in moderate to high load situations.

Suppose we relax our hypothetical conditions of high priority for NSW transaction processing and private memory for NSW swapping. Under more equitable sharing of system resources among a normal complement of users for a given host configuration NSW response characteristics will degrade beyond the values recorded in as previously table 2). (For example, see table 4.) Unfortunately, NSW memory demands seem to grow rapidly with increased simultaneous use. This would indicate the limited effectiveness of memory sharing, thus making an effort to identify an NSW "Performance Kernel" for special handling less attractive and not likely to succeed. The memory sharing behavior of NSW configurations is being investigated further.

7. What User's See Now

There is currently an NSW "user" system generally available to a limited user community. The NSW configuration is based around the ISIE TOPS-20 host, which is roughly the equivalent size of the BBND system discussed earlier. The ISIE host provides a self-contained NSW service (i.e., runs instances of all of the major NSW components), in addition to supporting remote tool bearing and front end hosts. The following data is derived from actual use of that system by unknown NSW users under random host load conditions prevailing at the time of use. Our only indication on the level of competition for system resources during these trials is the system load average at the time of the sample, which we include with our data. These results indicate the type of responsiveness which a current user would be likely to see under various levels of system load for a typical local host NSW transaction. To make comparison with previous real time responsiveness measures more meaningful, we have chosen to focus on the results of the RUNTOOL scenario. This is because with RUNTOOL there is no potential for a user dialog intervening within the transaction. Such a dialog is often required for file name disambiguation or confirmation, and may make complete response time measures for these scenarios less meaningful because of variable user think time. While it is certainly possible to establish different response time definitions for those transactions

involving embedded user interaction, we do not do so here to make comparison with previous experiments more meaningful. The table reports the approximate response time (in seconds) for various categories of system load.(1) For each load average interval, we report the number of trials sampled, the minimum observed response time, the maximum response time, and the average response time of the trials for that interval. The line indicated by "total" reports minimum, maximum, and average response time over all trials.

<u>load average interval</u>	<u># trials</u>	<u>minimum response</u>	<u>maximum</u>	<u>average</u>
0 - .9	9	5.4	11.4	8.0
1 - 1.9	4	10.4	20.6	13.4
2 - 2.9	10	10.4	23.8	16.6
3 - 3.9	2	13.4	27.8	20.6
4 - 4.9	3	25.4	33.2	29.4
5 - 5.9	5	18.0	34.4	24.2
6 - 6.9	2	31.2	31.8	31.5
7 - 7.9	1	21.4	21.4	21.4
8 - 12	3	35.4	39.0	37.4
Total	39	5.4	39.0	18.1

Table 4: RUNTOOL Response times (in seconds) under random conditions for the ISIE NSW "USER" system; local host FE, WM, and FM.

(1) The measurements actually recorded report the real time interval for a fixed part of the RUNTOOL scenario, not the whole transaction. This table reflects a uniform additional factor, based on knowledge of the transaction and previous measurement experience, to approximate the probable response time for the entire transaction.

8. A Closer Look at Resource Demand

From the preceding discussions, it should be clear that NSW transactions represent an extremely heavy demand on available system resources for all host configurations. Only the large configurations are able to adequately handle this demand, and then only for relatively few users (compared to "normal" usage patterns) before queueing delays drive response times quite high. In this section, we take a closer look at a major resource demand (processing time) for a sample transaction (GET FILE) to try to understand the heavy demand. We assume that the other major resource demand, virtual memory pages, is directly related to the large processing demand.

For the RADC-old system, the following table breaks down the per-component CPU demand for the participants in a single GET FILE scenario. This CPU time is for an unambiguous file reference requiring no inter-host message traffic. The net CPU column represents approximate CPU demand associated with the NSW implementation of that function, and does not account for secondary CPU demand for overhead items such as supporting the virtual memory. CPU times are in millisecond units.

<u>component</u>	<u>net CPU</u>
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 5: Individual process CPU demand for a single NSW GET FILE Transaction.

The total NSW CPU demand on a 2040 (no cache) processor for the GET FILE operation is about 1.5 seconds. The approximate CPU demand for the comparable local host operation (i.e., GTJFN JSYS) is about 50 milliseconds. Of the 1.46 second total demand, about 26% 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 much higher than the cost of plain message passing. The messages themselves follow an NSW-wide standard encoding scheme which is foreign to the internal format of most host systems. Thus, the form of the messages and the data passed in them require encoding and

(1)
This is the approximate time to schedule NSW processes whenever they logically block (i.e., voluntarily relinquish control to another process). It does not account for other scheduler wakeups (e.g., process page faults). Overall scheduler CPU time is much greater than the net CPU reported above.

decoding by both sender and receiver for every message. Based on measurements of an assembly language encoding/decoding implementation we will assume each encodement or decodement takes about 10 milliseconds (likely higher for high level language implementation and also obviously dependent on the complexity of the message), adding another 80 ms to the linkage CPU cost. There are also more subtle costs of this encodement for the self-contained host case. For example, the code to implement the function in each component, although conceptually the same, requires separate memory pages in the address space of each component. Even for those components written in a common language (e.g., WM,FP) and executing identical code sequences, there is no sharing of the code segments, which are built into each load module. 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. [The same is true for the Foreman during RUNTOOL.] 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, we see at least 580 milliseconds, or 38% of the processing demand associated with linkage. It is also clear from table 4 that the WM CPU demand of

almost 500ms (32% of total) for file name lookup, access control and related functions, and the FP CPU demand of about 450ms (29% of total) for file copying and related functions, represent significant demand relative to the cost of the OPEN File operation on the native operating system. An analysis of DELIVER file would be similar to GET FILE. For RUNTOOL and TERMINATE, the File Package is not involved at all; however, the Foreman and Front End assume more extensive roles in its place, leading to similar comparisons.]

9. Why?

In the previous section, we saw how the various components of the NSW structure consumed resources for a typical NSW transaction (GET FILE). In this section we examine some of the reasons for such a large processing demand. Because no performance data is available on the detailed implementation of many of the components, the explanations presented here are somewhat more subjective than in previous sections, and may represent only one perspective on the current situation.

Access to a file system member within NSW and within a native host operating system requires basically the same functions. First, control is transferred to a part of the system which gathers the appropriate parameters. Then the name is typically transformed into a complete name (flexible defaults usually mean that users most often do not supply complete file names), which is looked up in a file catalog. The result of the processing to this point is that the user's right to access the file is verified, and parameters relating to the physical storage of the file are obtained. The remaining processing involves setting up the support for direct tool access to the file data, and signaling completion to the requesting process.

Since the NSW GETFILE transaction is functionally similar to the native host Openfile, why does one demand 1515 milliseconds CPU time while the other demands less than 50? Much of the answer obviously lies in analyzing the largest uses of processing time reported previously, namely linkage, File Package demand (or combined FM/FE demand if we were to focus on Runtool instead), and Works Manager demand. We believe that there are three different factors which at least partially explain these three extensive NSW CPU demands. 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.

For the purposes of this discussion, we need to explain the current NSW implementation which has each major component as a separate timesharing job. This is structurally very different from a native host operating system in which typically each major component is implemented as some sort of efficient, although sometimes very complex, subroutine linkage. The evolution of the NSW structure is a fairly complex relationship between design, the notion of a prototype implementation, and organizational convenience. The design attempted to handle the most complex

situation where a user connected to one machine, might be running a tool on another machine, which referenced a file on yet another machine. Under these assumptions, there was a physical as well as logical separation of the tool interface and access to the file copy. In the interest of a simplified design and implementation, the file name lookup and access control functions also were logically divorced from the other functions. Thus, a design evolved where one component knew everything about the tool interface, but nothing about the file system; another component owned all of the logical file system and access control data but none of the files; another component owned all of the files, but had no logical file manipulation capability or direct tool interface. Further, only a single implementation strategy was pursued which also 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 at machine 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 systems become an overhead benchmark for functionally equivalent services, care must be taken to avoid, if possible, a

design or implementation which conflicts with the underlying system. For the TENEX/TOPS-20 family of computer systems, crossing any process boundary is expensive relative to internal subroutine linkage commonly used to support aspects of the TOPS-20 abstract machine.

Thus a message passing oriented architecture based on the existing implementation of the concept of a process is not ideally suited for host systems like TOPS-20. Despite the added cost, there are situations where multiple process implementations of certain functions are unavoidable on many available hosts, especially if we continue to limit all development to programs that runs as application code. In areas such as protecting segments of code or data from each other, and avoiding the addressing and capability limitations enforced by the operating system in developing its abstraction of a process, application code must be less efficient than the operating system itself because it has fewer tools at its disposal. The current NSW system has a number of areas where multi-process implementations were forced by the supporting host structure. However, the major portion of the extensive linkage processing demand results from NSW structured decomposition, not from NSW as host system application code.

In our opinion, message passing remains a valid system structuring technique, even for local host implementations. The

difficulty encountered is often in mapping this structure onto hosts which do not support these concepts at sufficiently low levels, and in competing performance-wise with an optimized linkage facility which usually was designed as the basis for the host operating system implementation itself. No NSW host has been developed from such a vantage point. The key to understanding NSW performance lies partly in the logical design, but also partly in a mismatch of the implementation of that design with contemporary host operating systems. Given the primary goal of the use of existing software on these host systems and the many fixed points they imply, it may be more accurate to say that an NSW system is not designed, it is engineered.

If all of the linkage costs were eliminated, NSW transaction processing would still not compare favorably with the local host equivalent. Part of the reason is a functional difference between the NSW and 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 bits 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 reasons for the file copy system model include the notion that this mode of access is appropriate to many aspects of software production, and 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.

[If we were to look at the RUNTOOL scenario, similar analysis would focus on the cost of establishing per-tool invocation direct connections as additional overhead not normally associated with the comparable native host operation. This type of connection handling helps support the additional NSW functionality of multiple, simultaneous tool sessions (even for all local tools) by a single user, despite a single physical terminal.]

That leaves the Works Manager processing demand as the remaining major unanalyzed item. The Works Manager performs the catalog lookup and access control check for NSW objects (files, tools). Native host file access implementations provide similar functions, and they are already included in our 50 millisecond approximation to the cost of the local operation. As noted previously, the main difference we see between the WM 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 NSW access control check, although probably not as efficiently encoded as the host access control scheme, is nonetheless very simple. It requires checking for the subset relation, and is not likely to require extensive code or processing time under current usage patterns. The NSW file catalog is organized around an information retrieval system, and a key word retrieval scheme. Currently under this organization, no attempt is made to organize the physical storage associated with cataloged items, and the mechanics of name lookup effectively use the entire catalog for possible matching entries. An item in the catalog can be found with equal ease (or difficulty) using any part of its name. Inserting an item into the catalog involves updating tables associated with each part of the file name to allow it to be retrieved with any partial file name.

All this is in contrast to conventional filing systems which if they support flexible file naming often provide a context for optimized access to limited search domains which contain most likely references at the expense of more general name references. Unfortunately, the software developing the NSW catalogs was not until recently adequately instrumented to evaluate its performance under frequent usage patterns. It seems clear that extensive WM resource requirements are related to the extremely flexible data

organization of its internal tables. Further analysis is required to determine if internal optimization of data base organization and processing algorithms can reduce catalog processing times to acceptable levels.

10. An Approach to Extended Optimization

The potentially expensive nature of an NSW file operation involving multiple hosts and file transmission and transformation was recognized very early in the NSW design process. In an effort to allow certain types of file processing to proceed without need for NSW file catalog or file copying operations, the current NSW concept of tool workspace was formulated to include the possibility of extending the effective lifetime of a file copy to coincide with the duration of the tool session. In addition, the concept that names for new workspace files were not disambiguated until delivery time allowed new file creation to proceed without NSW file catalog interaction. These concepts of tools operating in NSW provided a sort of caching of file copies within the tool workspace. When a file is referenced during a tool session, the workspace is first searched for an active file copy, and only when none is available is the NSW file catalog searched. Although these concepts are not fully and uniformly developed in the current implementation, they nonetheless have provided significant performance advantages for certain patterns of tool behavior, at the expense of a slightly more complex user model of NSW.

The reason for the discussion of workspace copies in this context is that it illustrates the feasibility of providing an alternative file system for tools integrated into NSW without

greatly increasing the cost of the operation over the native host environment when the relevant data is locally accessible. Currently workspace file activity does not provide automatic copying as in the NSW file system, nor does it provide any access control beyond isolation from the rest of the system. However, it does provide syntactic name transformation to provide name uniformity across host boundaries, and provides a uniform tool interface for interaction with remote NSW components when an operation cannot be completed strictly with local data. For the CPU overhead of a few milliseconds to obtain control and read parameters before the system call is processed by the operating system, and another few milliseconds to search a simple data structure representing only the workspace files, we can achieve low cost tool integration with local members of an alternative file system on TOPS-20 and hosts with similar runtime support structures. The operation becomes expensive only when the particular referenced data items require it to be.

The key concept is locally accessible data. The current NSW functional partitioning severely limits the scope and duration of data available to the components closest to the tools and the user for supporting independent processing. This results in the generally unacceptable system performance because process, job and host boundaries must be crossed too frequently, specific operating contexts are not maintained for any prolonged period and relatively

little accomplished is each single major transaction. The understanding of these issues unfortunately was only focused the hard way through a prototype implementation. However, the concepts of providing efficient access to available local data and providing for expanded sets of locally stored data both are playing a prominent role in the ongoing efforts to extend NSW functionality and improve its performance characteristics.

DAT
FILM
20

