NASA-TM-86089 19840020466

A Reproduced Copy

OF

NASA TM-86089

Reproduced for NASA

by the

NASA **Scientific and Technical Information Facility**

FFNo 672 Aug 65

NASA

Technical Memorandum 86089

CONSIDERATIONS ON COMMAND AND RESPONSE LANGUAGE FEATURES FOR A NETWORK OF HETEROGENEOUS AUTONOMOUS COMPUTERS

Norman Engelberg
Charles Shaw III

JANUARY 31, 1984

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt Maryland 20771

CONSIDERATIONS ON COMMAND AND RESPONSE LANGUAGE FEATURES FOR A NETWORK OF HETEROGENEOUS AUTONOMOUS COMPUTERS

January 31, 1984

Norman Engelberg
Charles Shaw III

Century Computing, Incorporated
8101 Sandy Spring Road
Laurel, Maryland 20707

for

NASA Goddard Space Flight Center
Contract NAS5-27197

ACKNOWLEDGEMENTS

The authors wish to thank the members of the TAE design and development team--Carmen Ana Emmanuelli, David Howell, and Dorothy Perkins of NASA Goddard Code 933; Elfrieda Harris of Science Applications Research; and Paul Butterfield, John McBeth, Philip Miller, Dharitri Misra, and Lora Mong of Century Computing, Incorporated-- for their help in producing this document.

The work for this paper was done under NASA contract NAS5-27197.

ABSTRACT

The authors consider the design of a uniform command language to be used in a local area network of heterogeneous, autonomous nodes.

After examining the major characteristics of such a network, and after considering the profile of a scientist using the computers on the net as an investigative aid, the authors derive a set of reasonable requirements for the command language.

Taking into account the possible inefficiencies in implementing a guest-layered net-ork operating system and command language on a heterogeneous net, the authors examine command language naming, process/procedure invocation, parameter acquisition, help and response facilities, and other features found in single-node command languages, and conclude that some features may extend simply to the network case, others extend after some restrictions are imposed, and still others require modifications. In addition, the authors note that some requirements considered "reasonable"--user accounting reports, for example--demand further study before they can be efficiently implemented on a network of the sort described.

1.0 INTRODUCTION 1

2.0 DEFINITIONS 2

3.0 NETWORK CHARACTERISTICS 3

4.0 DESCRIPTION OF USERS 5

5.0 USER MODEL/USER VIEW 7

6.0 REQUIREMENTS 11

7.0 NETWORK COMMAND LANGUAGE ISSUES 15

7.1 Description of TCL 16

7.2 Network Operating System Model 18

7.3 Command Language Features 21

8.0 SUMMARY 51

9.0 REFERENCES 52

APPENDIX A CHECKLIST FOR NETWORK COMMAND LANGUAGES

1.0 INTRODUCTION

With the increased popularity of local area networks (LANs), we have seen the development of networks of independently administered nodes, based on computer systems from different vendors. Along with the development of these networks, a need has arisen for a uniform command language through which applications users may manipulate a set of dispersed and independently managed resources.

Ideally, the command language is simply some universally known command language available on all the nodes on the net. Work on such standardized languages--languages that are the same no matter the underlying computer architecture or operating system--is ongoing ([ANSI/09SD 84], [COSCL 82]), but these languages are designed to provide a required set of functions; the committee charters do not permit significant considerations of possible implementation problems. (See [ANSI/05SD 79], [COSCL 82].)

In this paper, we investigate one common network case, a local area network with heterogeneous, autonomous nodes, in which the operating system the user sees--the "network operating system"--is layered on top of existing host operating systems, that is, "guest-layered". (See [ROBINSON 77], [MAMRAK 83], [PEEBLES 80], and [LANTZ 82] for existing examples of such a case.) This paper examines classic command language features to determine how the command language might be extended to help the network user, and to determine how the nature of a network of heterogeneous, autonomous nodes might make the implementation of a given feature impractical.

To perform this investigation we first examine the significant characteristics of a LAN with heterogeneous, autonomous nodes. Because we are investigating a user interface, we then describe the characteristics of the type of user for whom we are designing the interface and, based on the net and user characteristics, we propose a "user view model" (that is, the view of the system which we intend the users to have). Using this user view and keeping in mind the network characteristics, we describe a set of requirements on the command language. Finally, we present the features of a strawman command language, discussing the viability of the features in terms of the difficulties each feature may present to the implementation of the layered support. As a basis for our discussion we use an existing, host-independent, single-node command language, "TCL", which the authors helped design.

2.0 DEFINITIONS

In order to avoid some confusion in the discussion of the issues, we first present a short list of our definitions for some common terms and acronyms.

- node - a computer with memory and external devices
- home node - the node on the network onto which the user has formally logged on.
- remote node - any node except the home node
- local user - a user is local with respect to a node if the node is his home node
- remote user - a user is remote with respect to a node if his home node is any other node
- NOS - Network Operating System. The software provided to allow a user to access the resources of autonomous machines on a network. (See [FORSDICK 78] for an alternate definition.)
- NOSCL - Network Operating System Command Language, the object of this paper
- proc - a command procedure or an operating system process
- TAE - The Transportable Applications Executive, a user interface and applications support executive in use at NASA's Goddard Space Flight Center.
- TCL - The TAE Command Language, the reference language used in tnis paper; the command language for TAE

3.0 NETWORK CHARACTERISTICS

The type of network that concerns us has the following significant characteristics:

- The reason for the existence of the network is to share resources among the users associated with the different administrative centers; that is, the net is not primarily a testbed for distributed processing.

- The network is composed of heterogeneous nodes. Unlike some interesting existing nets used for distributed computing ([POPEK 81], [JONES 79], [LAZOWSKA 81]), we can make no assumptions about uniformity of the underlying computer architectures or the host operating systems. The node may be a mainframe running OS/MVS, a sixteen bit minicomputer with a memory-resident operating system, or a microcomputer-based workstation. (We do not necessarily include micros with arbitrarily small memories.)

- It is a local area network. We assume that the data rate between two nodes is on the order of between one and 10 megabits per second. (Note: there may be low-speed interfaces to other, possibly geographically distributed nodes; we do not want to exclude such an interface, but we will not drive the design of the command language by them.)

- The nodes on the network are under autonomous administrative control and the network operating system is guest-layered, that is, it is layered on top of the existing host OS's.

- The development of the capabilities of the net and the NOS are evolutionary; we want to be able to start with a set of basic capabilities and add to them as budgets permit and technology progresses.

- The network is loosely coupled; we cannot assume that any two nodes share memory.

The important implications we derive from these network characteristics are:

- Because of the data rates, we assume that it may be reasonable to process records from a small file by copying the entire file from one node to another, but it is not reasonable to copy large files (not, for example, a sixty megabyte spacecraft image file).

- A node owner may take down a node withrut consulting the network users or other node owners.

- A node owner may change the configuration of peripherals on a node, changing the address of a tape drive, or removing a line printer, for example.

- We can make no assumption about the existence or correctness of a user-readable clock.

- Not all users on a node will be users of the network operating system; a given node will support NOS users and users not concerned with the NOS or the network. Files may be generated outside the NOS, and accessed by both NOS and non-NOS users.

- Most common devices (most line printers, for example) cannot be exclusively allocated to the NOS.

- We cannot change the host OS to accommodate more comfortably the NOS or the NOSCL.

- Because of differences in computer architectures, and because we want to be able ro run programs that may use host operating system services, we cannot assume that the NOS can redistribute the application workload by moving an arbitrary process from one node to another.

- Although we can assume that the software providing the various levels of support for the NOS may be centrally developed, the only assumption we make about the release level of the software at a given node is that it is "compatible".

See [ROBINSON 77], [FLETCHER 82], [PEEBLES 80], and [LANTZ 80] for descriptions of nets that share most of these characteristics.

4.0 DESCRIPTION OF USERS

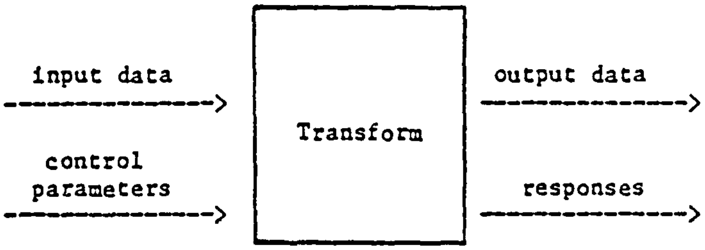
Current wisdom (e.g., [SCHNEIDER 82], [SMITH 82]) dictates that when one designs a user interface, one should be aware of the sort of task that the user is required to perform.

The following is a fair description of one common type of computer user; it is also a rough profile of the intended user of the TAE, the system for which the TCL command language was developed.

- o Someone with frequent need to use a computer as an aid in performing some analytical task.

- o Not a computer neophyte.

- o Has the following basic requirement, based on a "function" model:



  The user has a task to do; the task transforms the inputs into some outputs. We add to this basic requirement a recognition that the user has to supply some parameters, and that there will be responses generated in the process of exercising the transform. Thus the primary user requirement is to specify the location of the input data, the transform to apply to the input data, where to put the output data, and the parameters used to control the behavior of the transform.

- o Professional and cooperative, will not attempt malicious destruction of other users' resources.

- o Generally has a home base, that is, a computer most often used as the home node, but also has a need to log into any other arbitrary node.

- o Usually acquires and maintains files at the home node, often by using a process outside of the NOS (a realtime data acquisition process, for example).

- Finally, although the user's interface to standard software is through the NOSCL, the user may be satisfied with local resources, that is, the user may not want to use or be exposed to the network.

Two implications of the last characteristic are that a user on a node in the net should not suffer a degradation in apparent task performance as a result of the imposition of network software, and that the addressing of local resources should be the same as in the single-node case.

5.0 USER MODEL/USER VIEW

A user-view model is the model the user inevitably develops while using the system. Based on the apparent system characteristics and a level of abstraction that is comfortable to the user, it allows a user to predict what the system will do in a given situation.

By forming a user model before the user interface is designed we present ourselves with a target against which we may judge our design.

In this section we propose a user-view model of the network, based on the net and user characteristics described above. We assume the user corresponds to the "intermediate" level of sophistication, described by Schneider in [SCHNEIDER 82]. (See [HARDY 82] for why orienting a design toward the intermediate-level user is reasonable.)

The user logs into a particular node using a host-dependent login sequence. From this node, the user sees a collection of resources (processes, command procedures, files, devices) attached to autonomous computer facilities, each facility having a central computer and peripherals. The user manipulates these resources as necessary to perform a given task. The user can, for example, execute a proc on node #1 using a file on node #2 as input, and depositing the output in a file on node #3.

The user executes commands through a single uniform command language, implemented in the NOS command interpreter. In addition, a user may invoke user-written processes or command procedures living on any machine on the net. The user specifies which node to use in executing a process or procedure (GODDARD/COPY, for example).

A user may access files on any node in the net (given appropriate permission) by specifying the node name and the name of the file on that node. Devices on the net may be accessed by specifying the node name and the device, or generically, by specifying a suitable device type.

The user view is node-centered, that is, the user is registered as a user on a given node; if the user wants to establish a session by logging into another node, the user must be registered on that node (or use a guest account).

Most command error messages are machine-independent; some error messages have a machine-independent description with machine-dependent details.

This view contrasts with the view implemented on other systems in which the user need never be aware of the location of resources (e.g., [ROBINSON 77], [LANTZ 82], [FLETCHER 82]). In the proposed view, the user must know on which node files are located, is occasionally exposed to possibly confusing error message details, and receives a different view of the available resources depending on the node which is used as the home node.

We justify this view as follows:

- If the user knows that a node is about to go down (for example, for scheduled maintenance) the user may transfer files to another node.

- If the user knows that a node has gone down, the user may conclude something about additional files, devices, command procedures, and processes.

- As noted by Clark and Svobodova [CLARK 80] an implication of autonomy is that a user tends to use one machine and wants the access to that user's data to be most efficient from that machine.

- The user may configure his files such that important files are backed up on physically independent nodes.

- The user may want to control and display the costs of working on a network. For example, the use of a data file on the user's local node is apt to be cheaper than the use of a file with the same contents on a remote node; the cost of using functionally identical application processes on different nodes will differ according to whether or not the required data files are on the same node as the process, and on the speed and billing algorithms for that node.

- The user may wish to communicate with the remote operator; for tape mounts, for example, or for complaints, and information about computer downtime.

- If a user's files are not replicated, file access times will vary according to the hosting node; the user may want to plan file usage based on ease of access to some files.

- Some operations will not be available between two arbitrary nodes. For example, it may not be possible, in initial implementations of the network, for a user on a PDP-11 to access a file on an IBM machine.

- A user may need to perform more complicated recovery and detection if an error is due to a remote problem. If the user knows that the reference is local, then these procedures are not necessary. (This was cited by Clark and Svobodova [CLARK 80] for application programs, in which the program would have to provide logic for the more complex case if it didn't know that the reference was local.)

- Errors may occur as a result of the inability of the NOS designers to perfectly map the network OS onto the host OS, exceeding a quota or violating a privilege, for example. Although the NOS can map these errors into NOSCL errors, the user may need to know the host error code in order to correct the situation.

- The user may want to be sure that secure information is placed on a specific node.

- Although it is possible to design a user interface that has apparent network transparency, the user inevitably becomes aware of and must deal with the network. Assuming a user interface designed to make the network transparent, factors that support this assertion are:

  - The user will notice that file access times vary considerably according to the file being accessed.

  - Because of node autonomy and the varying reliability of node hardware, resources attached to some nodes will be generally less available than resources attached to others.

  - The user will note operating system-dependent terminal characteristics, for example the ability to type ahead when addressing some processes but not others (from VAX/VMS to VAX/VMS but not from PDP-11/RSX to PDP-11/RSX, for example).

  - Error messages from application software will display host-dependent error codes.

- Users on one node will communicate with users on another node.

Note that some of the items cited above reflect what we believe are the inadequacies of current software technology to provide these functions efficiently in a network operating system. In particular, copying arbitrary files from one node to another, user cost minimization in a heterogeneous node environment, optimizations for relative location of files and processes, and automatic recovery, are not now available in systems with heterogeneous nodes and guest layering.

Other nets that preserve a view of node visibility are COCANET [ROWE 82] and UNIX-UNITED [BROWNBRIDGE 82].

6.0 REQUIREMENTS

In this section, we present a list of reasonable requirements for a command language designed to support tne network and user view we have described. We list only those requirements of interest in the network case; for a more extensive set of command language requirements, see [ANSI/05SD 79] and [ANSI/06SD 79]. Note also that some of hese requirements are more proper'v requirements on a network operating system; they do, however, affect the command language design. Finally, although we conside- these requirements reasonable from a user's view, we note in Section 7 that some of these requirements present difficulties in efficien. implementation.

1. Related to files and other resources .

   - support for uniform file specifications and device specifications independent of the underlying host operating system, with at least one level of directory

   - file management commands: make a copy of a file (i.e., COPY), rename a file, list a text file, list a directory, delete a file

   - manipulation of a file using wild cards for a name component

   - location of a file by attributes other than the name (a file "type", for example)

   - logical names: the ability to map a user-defined name into an existing object name or another logical name

   - on protection: consistent user view of protection for all files accessible through NOS file specifications; basic read/write/delete protections at least by "owner" and non-owner; ability to protect files from non-NOS users, and to share read access to files with non-NOS users; ability to set and determine protection of a specified file

   - ability to specify a host file in host specification format

2. Related to handling of processes and command procedures

   - named packages of commands (procedures), with parameters and language support (loops, conditionals, variables)

   - process/procedure status query

   - specifying a host node to use in executing a transform

   - ability to initiate concurrent processes and procedures

   - procedure/process abort, suspend

   - automatic action upon a specified type of error (that is, some basic exception handling)

   - ability to run programs that have not been designed to interface with the NOS

3. Crash/recovery

   - notification of crash of any node with which the user is communicating

   - restoracion of files to a known state (e.g., previous version or "locked" or "recovered")

   - survival of the user/NOS interface following a crash of any node except the user's home node

4. For dealing with machine costs:

   - a way to determine the costs (storage and time) of having invoked a process or procedure

   - query for cumulative machine costs since login

5. For sharing other users' resources and communication with other users:

   - the ability to locate and access other users' files

- file/device protection

- query for file ownership

- determine id's of other users currently logged onto the network (under the NOS)

- send/receive messages and mail

6. Requirements related to visibility of the nodes and the network

   - the ability to log on to a remote host from the home node

   - query for the network status

   - query for status of one or more nodes in the network

   - ability to communicate with a remote host operator (for example, mount a tape, or receive a message on host downtime)

   - downloading a remote host

7. Help and error handling/status reporting

   - notification of errors using a uniform, host-independent message structure

   - determination of error details and possible recovery actions including ability to get help on an underlying host-dependent error originating at a remote node

   - help displays on processes and command procedures, built-in commands, general how-to-work the system

   - help on characteristics of the network (to the depth required to support the user view) and on network problems

   - help on host-host incompatibilities (in general help inquiries and error messages, e.g., "Image file copy from PDP-11/RSX-11M to WANG VS2000 is not currently supported")

8. General user-oriented requirements

- If a user specifies an operation requiring only local resources it should not be more difficult to perform or take a perceptibly longer time than the same operation in a non-network environment

- Some consistent time base and consistent user-view of time, that is, the user's perception of the passage of time based on time displays from the system should be consistent no matter the node, and the use of time in time-related attributes for resources ("time of creation" for example) should be consistent.

- Host command escape (the ability to execute a command in the command language of the underlying host operating system), local host and specified remote host

- "Reasonable" and consistent response times where by "reasonable" we mean a response time comfortable for the user; see pages 228 through 232 in [SHNEIDERMAN 80] for a discussion on user psychology and response times.

7.0 NETWORK COMMAND LANGUAGE ISSUES

In this section we briefly describe our reference command language, TCL, and present our basic model for the network operating system. Then, using TCL as a reference language, we discuss the classic characteristics of command languages and how they might be affected by the requirements we've developed.

7.1 Description of TCL

As a basis for the following discussions on network-oriented extensions to command languages, this section describes TCL, a modern command language designed for a single-computer system.

The following overview is a description of the facilities of TCL pertinent to the NOSCL issues discussed below; a complete description of TCL can be found in [CENTURY 83a] and [CENTURY 83b].

The major purpose of TCL is to provide a language through which users of the Transportable Applications Executive ("TAE") may invoke and provide parameters to scientific analysis software.

In TCL, an application program linked to run under the host operating system is called a "process"; a file consisting of a sequence of TCL commands is a "procedure".

A command in TCL is a TCL intrinsic command, or the invocation of a TCL "proc". A proc is a procedure or a process; all procs are located in TAE libraries (corresponding, under VAX/VMS [DIGITAL 82a], to a file directory).

Intrinsic commands are commands built into the TCL interpreter (the TAE Monitor). These commands consist of commands that perform general utility functions ("DISPLAY" to display the value of a variable, for example), and language-support commands such as the IF and LOOP commands.

In TCL, a proc is invoked using only the proc name; there is no "RUN" or "EXECUTE" command. The proc definition file--the file containing the procedure or executable linked image and the proc parameter declarations--is located by the command line interpreter using an ordered search of TCL libraries (similar to the path search in UNIX). The list of libraries to search is established by the TCL user during the session. Libraries are mapped directly into the host file facilities; under VMS, for example, a library corresponds to a directory. Intrinsic commands pre-empt procs; that is, if a command name is found to be the name of an intrinsic command, then no proc search is done.

A command is said to have "proc invocation syntax" if the form of the command invocation is,

```
<command name> <parameter value list>
```

A process receives command line parameters by calling standardized TAE support subroutines; a procedure receives the parameters by declaring them as parameters in the procedure.

TCL also supports the following traditional features:

- typed variables, arithmetic and logical expressions, built-in functions, and a LET command for assignment;
- IF, LOOP, BREAK, and GOTO commands;
- exception handling support;
- on-line help and message facilities

7.2 Network Operating System Model

In order to provide a basis for discussing the possible problems in implementing the various command language features, in this section we describe the basic model we use for the network operating system. The model we have chosen is appropriate for our user-view model; it is based partly on the single-node TAE model [CENTURY 83a] and partly on the service model in [FLETCHER 80] and the implementation of DIGITAL's DECnet [DIGITAL 82b]. The discussion is limited to those aspects of the NOS that clearly affect the command language interface.

We suppose a network of nodes, where every node contains the necessary hardware and software communication support for layers through the session layer of the OSI reference model [ISO/OSI 82]. Each node hosts, in addition, an executive, a directory server, a file server, and a "listener". The executive interfaces with the user; it interprets the user's commands, determines the command parameter values, and passes the parameter values to the responsible execution module. It also determines the status of the command execution and forms appropriate status responses for the user.

A user is "registered" on each node through a user node registry. If local access is desired, the user's name alone is registered (e.g., "JOHN"); if remote access is desired then the node from which the access is attempted and the name are registered (e.g., "NASA/JOHN"). (This approach is similar to the UNIX-UNITED approach [BROWNBRIDGE 82].)

A command module may be built into the executive command, or it may be a command procedure contained in a procedure file, or it may be a process (a program in host-dependent "executable" format on disk). If it is a procedure or a process (a "proc"), there is an associated proc definition file that the executive accesses to determine the type, default value, and other attributes of the parameters. If a parameter has no default and no value is specified by the user, the executive will prompt the user for a value. The proc definition file may, depending on the implementation, be incorporated into the procedure or executable program.

The executive uses the directory server to determine the location of files. Because nodes in our user model are exposed to the user, there is one directory server per node, and it knows only about the files and devices located on the hosting node. The executive or subroutines in an application program direct the file request to the correct node by finding the node name in the file specification (or using a default node name).

In general, files are not copied from one node to another unless the user explicitly requests so; processes read and write local and remote file records.

When a request is made to open a file, the request is sent to the directory server on the appropriate node; the server returns to the requester a file identification string or a "not found" status, or an "access violation" status. The possessor of the file identification string, the executive for example, then uses the file server to access the file. If the request is for a file on the same node as the requester then the file server may be embodied in a combination of host OS file services and the subroutines required to abstract them. If the request is for a file on a node remote to the requester, then the requester communicates with the remote directory server (through the listener, see below), and a remote file server uses the host OS facilities to perform the necessary file access services. (In practice, the directory server and the file server may be combined in one process.)

The file identification string provides sufficient information for the file server to check access rights against the requested operation (similar to capabilities [DAVIES 81]).

The listener on each node is responsible for the initial interface with other nodes. When a file open is requested by a user remote to the listener's node, the listener either spawns a directory server or communicates with an existing directory server, which thereupon establishes communication with the requesting process.

If the request is for the execution of a remote procedure or process, the listener spawns a local copy of the executive. The spawned executive communicates with the originating executive, obtaining the necessary context for the requested operation, and executing the process or procedure. Furthermore, if the proc is a procedure, the executive is responsible for executing the commands in the procedure. For a discussion of other functions and problems with this remote execution model, see "Proc Invocation" below.

A remote proc communicates with the user through subroutines in the process, which communicate with the proc's executive, which, in turn, communicates with the user's home executive. The user's home executive talks to the user (similar to virtual terminals; See [LANTZ 79] for a discussion on virtual terminals.)

Note that, for simplicity, our model does not at this stage provide for replication of any user files; we leave file replication to the user. In addition, the model does not address protection of objects; we leave the investigation of a protection model appropriate for our user model, and the associated impact on the command language, for future study. (See [DAVIES 81] for a general discussion.)

7.3 Command Language Features

This section discusses the important features of a strawman command language we call NOSCL; most of the features discussed relate to a requirement listed in the "Requirements" section, above; some of the features are based on existing features in TCL. Note that, where necessary, we assume the command line syntax of TCL [CENTURY 83a].

The descriptive technique we use below is to first describe a hypothetical NOSCL feature, then discuss the feature.

1. Session context

   When a user logs onto the NOS, a "session" is started and a session context is established.

   In NOSCL, the context consists of:

   - user name
   - session id - a unique identifier for the duration of the current session
   - a string that constitutes the command line prompt
   - a set of logical names that map into file names (see "Names" below)
   - "current" directory - the default directory string to be used if an object name is not fully-qualified
   - "home" directory - the initial "current" directory upon login
   - current setting of session global variables
   - proc search list - the list of directories to search for a proc invoked with an unqualified proc name
   - a set of user and installation-defined commands - command string equivalences and abbreviations defined using the "define equivalent command string" command
   - devices reserved

- o the user's NOS identification
- o identification of all procs currently executing
- o name of log file and state of session logging

The context is available to the NOSCL user through a set of global variables; to display a context component, a user may use the standard NOSCL command for displaying variables, or for some frequently requested components, the SHOW command (e.g., for showing the library search order).

There exist commands to save context to a named file, and to restore context from a named file.

Comments

We try to keep the context as small as possible for two reasons. First, the context will require memory in the executive (or from some memory pool), and, second, in our model of the NOS, when a proc on a remote node is executed, the home executive sends the entire context to the remote executive.

Note that the second consideration may turn out to be unimportant, depending on where the bottleneck for remote proc initiation is, on the effective rate of data transmission between two application-level processes in the network, and on the efficiency of the data encoding. (Binary data, for example, may have to be encoded as ASCII characters.) We estimate, based on experience with TCL (which does not have logical names), that the session context as defined above may be as large as 10 kilobytes. Note also that some optimizations are possible; for example, once the home executive has sent the context to the remote executive, it may, on subsequent proc invocations on that node, send only changed portions of the context.

Session quotas, accounting information, and session privileges are important components of the session context on an operational system, but hard to define and implement in a distributed environment. We have omitted them pending more study on the necessary NOS structures.

In addition, the context may include the current position of a magnetic tape, but it is not clear that this would be meaningful for the user to observe, nor can it be easily restored in a new session.

The current time and date may also be considered part of the session context, because it qualifies the rest of the context. See "Time" below for further discussion on this component.

2. Login

NOSCL users are required to login to a host, using the appropriate host-dependent login sequence; host facilities are then used to automatically log the user into the NOS (using VMS's LOGIN.COM, for example). In addition, a user may establish another session on the same node or a remote node using the NOSCL LOGIN command, a standard host independent login. In that case, the user must be registered as an NOS user on the remote node.

Upon initiating a NOSCL session, a special login NOSCL command file (called the "node login" file) in a reserved NOS directory on the home node is executed, which, in turn, executes a login command file in the user's default directory. There is one node login file per node.

A user is registered with the NOS in an NOS user registry (using a utility not specified here). The registry specifies the user's NOS identification, and initial defaults (the initial default directory for example). The login process accesses the registry for a particular user by that user's login name and, for a remote login, by the user's node name.

Comments

Further investigation is required for a complete specification of the data structures in the registry.

The login process establishes the initial user context using the NOS registry and the login command files. The technique wherein the system and user login command files are invoked is currently used in TCL. The user login command file together with the registry records for a user constitute the user's profile.

Note that a user that establishes a session on node A appears to the NOS as a different user than the same user establishing a session on node B; the user login file that is executed is dependent on the node. User login names are not unique for the entire network, but a login name is required to be unique for a given node. This approach has the disadvantage that, depending on how protections are implemented, a user may not be able to access files he created while logged onto another node. (If the registry is logically unique, the NOS may be able to map the user node and name into a unique protection token, thereby avoiding the problem.)

Although the location of the registry is not specified, we establish the requirement that the information necessary for establishing a session on a given node is resident on that node. This requirement is derived from the requirement that the performance for a user using the NOS executive but not the network must not be significantly affected by the existence of the network. See [BIRRELL 82] for more on user registries.

Note that the facility to login on one node and start a session on another host is extremely useful; it means that if the user knows the host-dependent login sequence of the local node, that user can start a session on another node without knowing the host-dependent login sequence for that node.

Finally, we note that, in a mature operating system, the registry would certainly include the user's privileges and session quotas; it is not clear however, what the list of privileges should be on an NOS, nor do we yet understand how to handle quotas in a distributed system.

3. Names

In NOSCL, an object with a standard name is named by its node and by a path-name within a node. A fully-qualified name is of the form,

```
<node-name>//<directory>/.../<directory>/<simple-name>
```

If a name starts with "//", or "/", then the object is to be found on the local node, and the first directory after the "//" or "/" is the user's home directory. If the name does not start with a node name or a "//" or "/", then the current default directory string is assumed to precede the specified name. The first directory name in the string is the name of a user if the node has more than one user.

If the name is preceded by a to-be-designated special character, the name is taken as a name in host OS format on the home node; if the special character appears after the node marker, the name is taken to be a node name and the string following the special character is to be interpreted as a host file specification on the designated node. Host names must be enclosed in quotation marks if they contain any of the NOSCL special characters.*

The maximum length of a node-name, directory name, or simple-name is the same for all nodes, as is the maximum depth of the hierarchy.

The objects that have standard names are directories, devices, files, users, processes, and command procedures.

The fully-qualified name of a device on a given node is,

    <node-name>#DEVICE/<device-name>

For example,

    N1#DEVICE/LP10

The <device-name> for a given device is established by the system administrator for that node; conventions will be established for uniform naming by device type.

For all objects with standard names, except processes and procedures, if an object's name is not fully qualified, the name is formed by placing the current default directory string in front of the name specified.

For processes and procedures, there exists a search list; search lists are described below, under "Proc Invocation".

In addition to the names described above, a user may define a logical name. A logical name is a name that is defined to map into an object name or into another logical name. For example, a user may define TESTFILE

* A special character is any character having significance to the command line interpreter.

to map into /NHE/TESTDIR/TESTFILE, or LP to map into N1#DEVICE/LP10. There is one logical name directory per user, with initial entries typically set by the node and user login command files.

Comments

The fully-qualified name specification described above was chosen because it conforms with the proposed user-view model and because it is compatible with the latest specification issued by the ISO SC16 committee. [ISO/SC16/N1454 83] Note that, by having both the two distinct separators ("#" and "/" in our case), we allow a form of fully qualified name whereby the node name defaults to the local node. See [FLETCHER 82], [ROBINSON 77], and [LANTZ 80] for nets with heterogeneous computers and no node name in the file spec.

A possible extension to the name specification used by the ANSI committee is to provide for hierarchy of net names, N1/N2/N3#JOHN/AFILE, for example. (See [ANSI/09SD 84].)

Names for variables, exception handlers, and labels are not considered standard names as defined above. The names for these objects will depend on the procedure language; in TCL, they are simple names only.

Currently under investigation by the ANSI X3H1 committee is whether or not any object attributes--the VMS file "type" and version number, for example--should be part of the name. We will wait for further results from the committee.

A related question is whether or not to support access by object attribute when the attribute is not part of the name. (It may be another parameter in the command, for example.) This technique is particularly useful if device type is an attribute; a service can be defined to return the names of all objects with the device type attribute set to a specified string, "line printer", for example. Further progress on these issues will follow investigation into what attributes should be supported. We note, however, that search-by-attribute is an expensive mechanism even on one-node systems (generally requiring a separate set of pointers).

The technique for naming devices is taken from UNIXtm

[BELL 79], and has the disadvantage that it reserves a directory name. It is expected that installations will define logical names for all devices.

An alternate technique to be studied is allowing the device to take on any name in any directory and defining an attribute, as noted above, that indicates an object that is a device of a specified type [AGRAWALA 83]. This would allow a system manager to name the "device" directory by any name; a user would reference a device through a logical name or by requesting an object with the specified attribute.

The restriction that the maximum depth for directories is the same for all nodes is made for user-friendliness, but may use too much processor memory for nodes with little memory to spare. (The maximum depth of directories is an important number in determining memory use by the executive, since it determines the space allocated for file type parameters and variables, and for logical names.) For ease of implementation and portability, consideration should be given to limiting the depth of the directories to some fixed known value, as in GRAPEVINE [BIRRELL 82], and Clearinghouse [OPPEN 83].

Logical names provide for node name transparency, particularly useful for device names, aliases, and "standard" devices (standard output device, standard error device, etc.)

There are no logical names automatically known across the network, that is, no network-central directory of logical names; system administrators coordinate the node login command files to define common logical names for devices and files used by many users. (The assumption here is that the logical names used across the network, names that map into devices for example, change infrequently.) We have avoided centralized directories for logical names because they imply either a remote access to the central directory every time a name is referenced, or replicated directories. We have provided instead a primitive form of replication, that is, system administrators updating the node login files through an editor.

For flexibility, logical names are bound to the physical name when the name is used, and not before. Some of the tradeoffs on logical name binding, on network-wide logical names, and on directory replication are discussed in the GRAPEVINE and Clearinghouse papers cited above; Watson also discusses binding in [WATSON 81].

Open questions related to logical names are:

- Can a logical name map into a node name?
- Can it map into a node name plus the left part of a file name? (Consider LN#A.B where LN is defined to map into N1#X.)
- Should logical names be restricted to the left side of a name specification (as in VAX/VMS) or can any component of a name specification be a logical name?
- Should logical names have attributes? If the design is such that the user cannot tell a logical name from an object name, then logical names should have attributes. (This capability is useful when searching by attribute: the search would yield a list of object names and logical names.)

An issue not discussed above is in what contexts wild cards should be permitted. Clearly, on a network, a wild card in the wrong place can produce disastrous consequences.

Finally, we note that our syntax for host file names may result in awkward file specifications such as:

```
N1#%"[1,1]x.y"
```

assuming the percent character marks a host file specification.

4. Proc invocation

Process and command procedures are invoked by naming the proc and providing the values of its parameters. For example:

```
PROCP1 1, 2, 3
```

If the name of the proc is the fully-qualified name of the proc definition file then the proc definition file in the specified directory is used; otherwise the following proc search algorithm is used.

If the executive determines that a command is not an executive built-in command, the executive assumes that the command is implemented as a process or procedure. It then searches in the following directories, in order:

1. the user's current default directory

2. the directories in the search list established by the SETSEARCH command

3. a directory designated as the "system" directory on the local node

A search list cannot include a directory on a remote node; to invoke a proc on a remote node a user must fully qualify the proc specification, including the node name. Furthermore, if the currently executing proc is a procedure on a node remote to the user's home node, then the search list is automatically restricted to the user's current default directory, the directory of the remote procedure, and the system directory; that is, the variable part of the search list is reduced to the directory of the remote proc only.

If the user does not specify all the required parameters for a given proc, the user is prompted for the missing parameters. A parameter that has a default defined is not considered a required parameter.

Comments

The basic proc invocation syntax is, from the user's view-point, a common one. It is currently in use in TCL, and is similar to that being used in COSCL [COSCL 82] and in ANSI X3H1 discussions [ANSI/09SD 84]. The notion of prompting for user parameters is available in TCL, but the "tutor" mechanism is preferred. (See the TAE User's Manual [CENTURY 83a] and Programmer's Manual [CENTURY 83b] for details on this and on the specification of parameters on a proc invocation line.)

There are some problems with implementation of this simple model on a network of the type that concerns us.

In TCL, there exists for every process and procedure a definition file. (For a process it is in a text file distinct from the process; for procedures, it is located at the start of the text-formatted procedure.) The proc definition contains parameter descriptions, necessary for parameter value checking and for parameter prompting; a description includes the type of parameter, the parameter default value, and, possibly a list of valid values.

Given these parameter descriptions, we have the problem of how to efficiently prompt the user for parameters when the proc is located on a remote node. Using our NOS model, if a proc is on a remote node, an executive on the remote node handles the command line processing. If the remote executive does the prompting, we have a relatively slow interactive dialogue; if the local executive does the prompting then it must access a remote file (and probably copy at least the parameter description portion), thereby forcing a lengthy proc initiation time. Another approach is for the parameter descriptions to be located on all nodes that access the proc; this approach would probably be faster (it still requires both executives to open a file, but no remote access), but it presents a proc maintenance problem: the person responsible for the proc must remember to check for all duplicates of the file when updating it.

Listed below are some additional problems in regard to proc invocation:

- In the current TCL, the author of a proc can specify that a parameter is an "input file", that is, the executive is to check that the file exists. This implies the file must be opened twice: once by the executive, once by the proc. Should this capability be retained given that the open may require several remote file accesses through the NOS file system (which is layered on type of the host system)?

  An argument in favor of early checking for file existence is that it may be expensive to run the proc on a remote node, and then to discover during proc execution that the file doesn't exist.

  Note that it possible to generalize the early-check capability to all devices, that is, we can specify that a proc will not run unless all devices specified in the parameter description set are available. We prefer to let device availability be tested when the device is required.

- Because there will be configuration differences between copies of the executive on different nodes, a user might find that some proc invocations are acceptable on some nodes but not others. Thus, for example, a user may specify a long string value for a parameter and the string may be rejected (because the maximum string length on one node may be different than on his home node). This problem can be solved by declaring that there are no differences between executives, but such a solution is unsatisfactory when the range of processor memory size and disk space on the net is great.

(The entities that are parameterized in the current TCL executive include number of parameters allowed on a line; number of command line continuations; host filespec length; maximum length of message key; maximum allowed IF nesting in a procedure; maximum depth of proc nesting; maximum number of characters in a string parameter; and maximum number of values for a vector parameter.)

- If a user has defined a number of new command strings (defined "COPYIMG" to mean "COPY IMAGEFILE", for example), we have the following problem: a procedure is developed on node N using the command strings defined by both the system manager (in the node login file) and the proc developer; the context of the execution, however, contains only the command strings defined by the user and user's system manager. (For example, the procedure may assume COPYIMG is defined, but no such definition exists on the home node.) Using NICOLA/KIWINET's "abstract machine" concept ([EFE 83], [KUGLER 80]) we can say that a procedure is developed to run on a certain abstract machine, and that our model does not provide that machine.

- If NOSCL supports floating point parameters (as TCL does), then a problem seen on nets with heterogeneous nodes is that it is impossible for a parameter to maintain the same precision among different hosts. The precision in a user-supplied floating point number may be lost when converted to the node that hosts the proc; a proc that does nothing at all except return the input value, may, in effect, change the value. See [KIMBLETON 81] for more discussion on data transfer between different hosts.

In regard to proc search lists, we do not permit search lists to reference remote nodes because we believe that the increase in the maximum search time (incurred on mis-typed proc names) would be unacceptable to the user, and because the cost associated with the execution of a remote proc may be significantly greater than that for a local proc. Thus we force the user to specify the node name (or map to it using a logical name).

For procedures executing on a remote node, we set the variable part of the search list to the remote procedure's directory only, because the complete search list as seen by the user on his home node refers exclusively to directories on the home node, a node by definition remote to the node executing the procedure. Thus, if this restriction is not imposed, a proc search from a remote node would use only remote accesses. The remote procedure's directory is placed in the search list because access of this directory is local, and because proc packages often consist of several procs in one directory.

Note also that search lists on single-computer systems have been criticized [BEECH 80] as being unkind to the user in that a user may find that a command defined in one way one day is defined differently (that is, with another proc) another day or on another system. If these lists are made to span autonomous systems, which may use different naming conventions, the definition of a given command is still more unpredictable.

Two significant problems relate to the invocation of programs that were not designed with the NOS in mind. The first problem is that the program's parameter interface will not match the NOSCL interface; that is, the programs do not use the standard NOS "get parameter" routines. Assuming the program gets its inputs from a command line, approaches to this problem are:

- create a host-privileged program that captures the parameters in NOS-standard fashion and uses host operating system services to form the proper host-dependent inputs;

- translate the NOSCL command line into a command line acceptable to the host;

- provide, on each host, a procedure that performs the mapping

See [BRADEN 80], [LANTZ 80], and [KIMBLETON 78] for more discussion on this problem.

The second problem is that there will exist on several nodes on the net some utilities--a FORTRAN compiler, for example--that perform identical functions but which provide different options depending on the host operating system. The FORTRAN compiler on some--but not all--machines may provide a "debug" option, for example. Possible approaches are to define the NOSCL interface such that it provides all possible options; define the NOSCL such that it provides a useful subset; let the host machine provide a NOSCL procedure for each such utility to present the options relevant to that host; or to call it a host-dependent issue, and just let the users use a host escaping mechanism. Because no changes to the command language are required, we favor the procedure approach. See [SCHICKER 75] and [KIMBLETON 78] for more discussion on this issue; see also the standardized command language efforts ([COSCL 82], [ANSI/09SD 84]) which must confront this problem as well.

5. Protection

To perform an operation on a file (make a copy of it, for example), a user must have "appropriate" access rights.

The "create file" command provides the user with the capability to set the file protection, explicitly, or by default.

There are commands to change the protection of a file and display the protection of a file.

Comments

While noting that protection is more in the domain of NOS study than NOS command language study, and also noting that the business of mapping protections for a guest layered file system into a host system is an enormously complex topic, we nevertheless outline the characteristics that we believe appropriate for the proposed user-view model and requirements:

- The protections we refer to here are associated with files named using the NOS file system, not the host file system (although we assume that the NOS file system is layered on the host file system).

- We consider files and directories only; the system may or may not extend to other objects; at the user level it is not now clear what other objects need to be protected. (Devices here are considered files.)

- The basic requirement is for a simple system; a highly secure system (one that implies data encryption, for example) is not required.

- The basic protections required are:

  - Some users will have read, write, and delete access to a given file, according to the class of user, as described below.
  - Some users will have no access whatever to a given filc.
  - The "owner" is defined to always have all access rights (read, write, delete).
  - The owner of a file must be able to protect files from the class of users consisting of all users remote to the node on which the file exists.
  - A user must be able to protect files from the class of users consisting of non-NOS users, and must have the ability to share read access with non-NOS users.
  - A user must have the usual necessary access rights to directories to traverse a path ending in a file. (To delete a file, for example, one must have "write" access to the directory.)

- A final requirement: the protection system must not make file accesses unreasonably slow.

See [IFIP/WG2.7 1983] for a protection model relating to command languages.

6. Device management

The RESERVE command grants the user exclusive access to a specified device; the RELEASE command releases it.

The RESERVE request is not queued; if the device is reserved by another user, the request is denied.

There is also a command to show the status of a specified device. Because all devices are seen as files attached to the DEVICES directory on each node, the status of all devices on a given node can be displayed using a wild card with the same "show status" command used for a single device.

A possible status for a device, added for the network case, is "device not available to non-local users."

Comments

Note that we use exclusive reservation for simplicity. More refined levels are of course possible; for example, reservation such that the user is only guaranteed that no other user will write to the device.

The following questions remain to be investigated in regard to device reservation:

- Is it possible for a user to reserve a device that is accessible to non-NOS users? Assume we have an NOS process on each node called the ALLOCATOR, which controls allocation of the device among NOS processes. If the device is shared with non-NOS users on a node, ALLOCATOR for that node must somehow reserve the device under the host OS, then pass to the requesting user's executive the exclusive reservation such that it is recognized by the host OS.

- Do we provide generic allocation, that is, allocation of any device of a specified type? If so, what is the best mechanism? One mechanism, used by VAX/VMS, is to allow the user to specify the device name without the device number ("LP", for example, rather than "LPA0"). Another method is to make device type a parameter in the RESERVE command.

  A third approach would be a two-step method. Assuming device names are in a directory along with an associated "device type" attribute, we can suppose a "resource locator" service, which returns to the caller a list of names that satisfy a specified attribute. The user may then obtain the list and reserve one of the devices named therein (assuming it is still available). This method is appropriate if devices can have arbitrary names (see "Names" above) [AGRAWALA 83].

  A problem with the two-step method is that it requires two steps; a device may become unavailable between step one and step two. A variation on this method is to define a command language function that returns an identifier of the first available device of the specified type; the user may then, for example, "RESERVE fn('line printer')"

- We have provided no mechanism for avoiding deadlocks between two or more procedures contending for the same set of resources. For example, procedure P1 may reserve device D1, then wait on D2, while P2 has reserved D2 and is waiting on D1. Note, however, that the procedures would have to contain explicit retry loops (since there is no automatic wait).

  If explicit retry loops are used, we note that a remote user is less likely to be able to reserve a device than a local user. A requester local to the node on which the device is located is more apt to be successful simply because the request can be tried more often. In an extreme case, the remote user may have to wait on several users that requested the device long after the remote user. This is largely a consequence of our decision to not queue the request.

- Should there be a facility whereby a user or a proc is signalled when a device becomes available?

- If the device is reserved by a user on a node and the node crashes is the device automatically released?

In regard to the "device not available to the network" status, there are several reasons a device might be available on the local node, but not from a remote node: The device might require the attendance of the user; the device might be a "demand" device, that is, it may supply or require data faster than can be supplied across a network link; or the owner of the device may simply decide to not make it available.

A question to be investigated is whether or not the availability of such a device can be accommodated using the available protection mechanisms of the NOS.

7. Time

All times seen by the NOSCL user are local to the clock (if any) on the specified node.

There is a command to show the current time on the local node or on any specified remote node.

Files have a "time of creation" attribute; the time is local to the node on which the file resides.

Comments

We have said that to have a single time base from the user's point of view is a requirement. By this we mean that if the user requests the time from any node--if, for example, the user is logged onto a remote node--the time the user sees will be independent of the node; in addition, if a user U1 creates a file F1 on node N, and user U2, on a different home node, creates F2 at the same time that F1 was created, then both U1 and U2 should be able to display the file creation times and find that the file creation time for F1 is the same as the file creation time for F2 (plus or minus a few seconds).

There are difficulties in maintaining a uniform time base across the network:

Examples of the possible problems are:

- the clock on one node was never set at boot time (a problem for single-node systems as well);

- the clock on one node is defective, running at a significantly different rate than the other clocks in the net;

- a node is in a different time zone than another node;

- if we try to set up a single time across the system using messages, then there will be a special "NOS" time, distinct from the respective host times, a phenomenon likely to cause confusion for users and system administrators (but one evidently handled by computer users who schedule by Greenwich Mean Time).

Note that a useful capability that depends on a solution to the time base problem is the ability to run a proc at a specified time; the common time base is critical if a the initiation of procs in a series must be synchronized.

Possible approaches to resolving these problems are to be studied; see [LAMPORT 78], [REED 79] for helpful primitives.

8. Asynchronous procs

A user may invoke a proc, explicitly indicating that the proc is to run "asynchronously", that is, the proc is initiated, and the user is prompted for the next command while the proc is running.

When an asynchronous proc is invoked, the executive prints a message on the user's terminal indicating the name of the proc and a NOS-assigned proc identifier; in addition, if the user provides the name of a variable, the variable will receive the proc identifier.

There is a command to wait on a proc with a specified identifier, a command to display the status of a proc with a specified identifier, and a command to display the status of all active procs. In addition, the ABORT command aborts a proc with a specified identifier.

The WAIT and ABORT commands constitute synchronization commands; an exception handler may be entered only following execution of a WAIT or ABORT.

The following additional rules apply to an asynchronous proc:

- An asynchronous proc survives the termination of the invoking proc, but not the termination of the invoking session.

- An asynchronous proc can modify a global variable only if the global variable was defined to be SHAREABLE (see "Variables", below).

- An asynchronous proc cannot declare output parameters.

- Any changes or definitions to logical names, the "current" directory, the proc search list, or defined command strings have an effect only on the asynchronously executing proc and its children; the changes do not affect the parent hierarchy.

- An asynchronous proc cannot reserve a device.

Comments

The ability to execute computations concurrently is important for efficient execution of a distributed algorithm. Work has been done for several years on the most efficient methods for controlling and synchronizing concurrent operations. (See, for examples, [LISKOV 79], [ICHBIAH 79], [HOARE 78], [REED 79].)

The philosophy we have chosen--to provide the command language user with the ability to run a proc asynchronously and to determine when it is done--is the current TCL philosophy. We do not provide any significant resource synchronization mechanisms, nor any inter-proc communication mechanisms, leaving those capabilities to established programming languages.

Furthermore, we have restricted the operation of asynchronous procs such that we do not have to provide mechanisms for serializing access to session context data. By our proc invocation model, when a remote proc is initiated, a spawned executive receives a copy of the context data; thus we have our session context data replicated, possibly on more than one node. The restrictions we impose seek to serialize access to the context data by ensuring that only procs run synchronously can change the parent context.

We do not permit an asynchronous proc to have output parameters because output parameters must refer to a proc variable, whereas we do not require a parent proc to outlive a child asynchronous proc.

We have declared that an asynchronous proc does not survive the death of the parent user session. We therefore cannot conveniently support applications such as a monitoring system in which a user dispatches several asynchronous procs on different nodes and then logs out, nor can we provide the user with the capability to initiate a proc on another node, knowing that the home node is soon to be taken offline.

We have taken this approach because it is simple, giving a user adequate and safe control. The following topics must be addressed if we allow these "detached" procs:

- Noting that a detached proc can run "forever", using resources on many nodes, should some sort of privilege be required to initiate a detached proc?

- Should a proc be allowed to initiate a detached proc without explicit user permission?

- How must our proc invocation model change to accommodate detached procs? (If, for example, a detached proc wishes to conduct an interactive dialogue, how is it handled?)

- Assuming remote batch is supported, how many of the capabilities provided by detached procs are also covered by remote batch?

- Should an exception handler in a detached proc be executed when the proc that initiated it terminates?

9. Variables

"Local" variables are local in scope to the NOSCL procedure in which they are declared. Local variables may be defined by an interactive user; they are then accessible only by that interactive user.

"Global" variables are accessible from any procedure or process (remote or local) invoked within a session of a given user, that is, they are defined for the single user. The user at the interactive level or any synchronous proc (with an appropriate declaration), may read or write any global variable; an asynchronous proc may read any global variable, and may read or write any global variable declared SHAREABLE.

A synchronous proc may also define "output" parameters, however, we do not state whether the variables into which the output values are placed are set during proc execution (call-by-reference) or upon proc termination (call-by-value-result).

Comment

This is the current TCL approach, extended by the SHAREABLE case.

We do not provide in the language variables that are global to more than one user. Such a facility might be helpful for inter-procedure semaphores, however, we leave inter-proc communication to processes, which have available well-known inter-process communication facilities.

The restriction that asynchronous procs may have read-only access to a global unless it is declared SHAREABLE is set to avoid unexpected asynchronous effects; this is similar to the single-node case in classic programming languages where a user must place a variable in a special place in memory (a designated FORTRAN COMMON area, for example) if it is to be shared. Note, however, that if shareable variables are provided, we may need to provide a transaction mechanism, because we then have the possibility that there is a dependency among two or more shareable variables--variable X is twice Y, for example. A crash of a remote proc after updating one of the variables, but not the other, would leave the variables in an inconsistent state. See [LAMPSON 81] for more discussion.

10. User attention sequence and proc aborts

An attention sequence is defined for each host OS (control-C, for example on VAX/VMS). The attention sequence solicits the attention of the local executive.

When the attention sequence is signalled to the executive, the executive suspends execution of the synchronous proc; the user may then abort a proc, suspend a proc, continue, or execute a built-in command that displays status or help information.

If the attention sequence is used in order to get the executive's attention while no synchronous proc is running (to break through a terminal read by an asynchronous proc), then all executive commands are available, including proc initiation.

If a procedure is aborted, any nest of procs below that procedure is aborted as well.

Responses from a proc may appear at the user's terminal after the proc has been apparently aborted.

Comments

The synchronous proc is automatically suspended to allow the user to immediately stop an operation out of control. Generally, users will run remote procs asynchronously; remote procs cannot be automatically suspended because the executive doesn't know which proc to suspend. Note that, when the user enters the attention sequence, the user must suffer the communication delay required to send a "suspend" message from the local executive to the remote executive of the synchronous proc.

Messages may be written to the user terminal after apparent proc abort because of communication delays in sending the abort message to the remote executive.

The user is restricted in the domain of operations during the interrupt period because of the distributed session context problem; it is not clear how the context for a synchronous remote proc would be affected if the user is able to execute commands that change the context.

Some open questions to be studied:

- If an aborted procedure has initiated asynchronous procs, are any procs it has initiated asynchronously aborted as well?

- If we permit remote synchronous procs to run additional synchronous remote procs, does an attention sequence result in a cascade of "suspend" messages from parent to child?

11. Crashes

If a user has invoked a remote proc (synchronous or asynchronous) and the node on which the proc executes crashes, the user is informed that the proc has failed because the node has crashed. The node name is provided as well.

If a node crashes and it has initiated other procs, those procs will be aborted.

Comments

We assume that the communication software in the machine that invoked the proc has the capability to detect the crash of a node hosting a proc it has invoked. The crash would be signalled to the invcker through the normal proc termination mechanism, and the detailed status would indicate "crash of remote node...."

More analysis is required in regard to crashes. Some of the questions are:

- o Are there any "atomic actions" of concern to the command language, that is, are there any command level user actions that must be rolled back to a synchronization point? (See [LAMPSON 81] and [LISKOV 81] for a discussion on atomic actions.)

- o We have said that procs do not survive a crash of the invoking proc's node. The reasons for this decision are:

  - Procs are intended to be agents of an interactive user; batch should be used otherwise (see also discussion on "detached" procs, under "Asynchronous Procs," above)

  - If procs are allowed to survive, then a user logging in would have to deal with the possibility that there are outstanding procs that belong to his session.

  - We could provide a session recovery mechanism, so that the parent session and proc is restored to some synchronization point, but the recovery and re-establishment of the communication to the proper point is too complex to be worthwhile in a command language, requiring transaction processing mechanisms and delicate timers in the surviving procs. (See [LAMPSON 81] for a discussion.)

  Assuming that the proc is not aborted, where do responses go? Are they saved in a file and sent when the crashed node recovers?

- o If the parent node crashes and the child proc is aborted should an exception handler in the child proc be triggered? This would be a useful, cleanup mechanism, however, because the exception handler could continue as if nothing happened, it implies that the NOS must have a timeout mechanism on the exception handler.

- o Is there an efficient way to determine how much a crashed remote proc has accomplished? The problem is that some sort of logging is required: if the log is on the same node as the remote proc, the user cannot make the determination until that node comes up; if the log is on a node remote to the logging node, every command to be logged would require a remote access. A possibility is to make this sort of logging optional, thereby making it available for long-running procs, or for procs that run on unreliable nodes.

o From [SALTZER 78]: How does the system manager know when a node can be brought down without interrupting an active remote operation?

12. Procedures

The full command language available for procedures on single node systems is available on multi-node systems; furthermore, all commands are available to the procedure regardless of whether the procedure was invcked by a user on the same node as the procedure or by a user remote to the procedure.

Comments

We make this assertion for want of any apparent exceptions.

In addition, we offer the following notes on factors affecting procedure portability:

- Generally, we cannot assume that integer overflow occurs at the same value on all nodes on the system. Integer overflow at the same value is user-friendly, but it is difficult to pick a suitable integer size and inefficient to implement the same size integer on machines with word sizes from sixteen bits to sixty bits.

- If the procedure language provides a floating point type, the definition of a floating point number will vary from node to node.

- As noted under "Proc Invocation" above, a proc may have been developed assuming a given set of user-defined commands.

- Inevitably, some procedures and processes will use host facilities, generally for efficiency, often because the facilities are not made available by the NOS.

13. Logging/History lists

NOSCL supports a session history consisting of interactive commands to a single log file. Commands from procedures are not logged.

Comment

TCL currently supports this level of logging; procedure commands are not logged because the TCL designers felt that the log would become too large too quickly.

If logging of commands in remote procedures is to be considered, the problem of merging the logging of commands from the home node with commands from remote nodes must be studied. Note that commands from any procedure may be logged to the standard output file, if output is redirected appropriately.

14. Exception handling

Exception handling follows the model used in TCL: A procedure may designate that, upon some "bad" status from a proc or command, a procedure-defined "onfail" command is executed (typically a GOTO). There is no retry capability and no distinct exception signalling. See [CENTURY 83a] for a description of the TCL model.

In addition, we specify that an exception from an asynchronous proc is signalled only after a synchronization command. See "Asynchronous Procs," above.

Finally, we also define for the NOS a set of standard NOS status codes for all known errors. (Thus, a procedure exception handler can check for "node down" status and perform appropriate recovery.)

Comment

We see no problem with this model in regard to working in our network environment.

A problem to be studied is whether or not an exception should be triggered upon the crash of a remote node that hosts a currently executing proc, or upon the crash of any user-specified remote node. Our current position is that first case will be handled by normal proc termination handling; the second case is not handled. Note that, for a command language that supports exception handlers at the interactive level (TCL does not), a "node crash" exception can be the vehicle for printing an appropriate message to the user, or possibly doing an automatic reconfiguration.

See [LANTZ 80] for more discussion on exception handlers in distributed systems.

15. Help facilities

Help is available on built-in commands, on procs (remote and local), on proc parameters, on error response details, on the network configuration, and on node-dependent capabilities and parameters.

The help command also has a variation whereby the user can obtain help on a built-in command as implemented on a specified node.

Comments

The help information on built-in commands is made available when a new version of the executive is released. The user is provided with the ability to direct a help request on a built-in command to a specific node because there may be different versions of the executive on different nodes. (Recall that the remote executive executes procs submitted to that node.)

The help information on node-dependent capabilities is made available when the executive is ported to a given node. Included in this category are:

- o the computer and operating system installed at this node;
- o the version number of the NOS executive installed at this node;
- o the values for executive configuration parameters (the number of parameters allowed on one line, for example);
- o the values for host-dependent parameters (e.g., the highest integer allowed);

- o host-dependent restrictions (e.g., no floating-point)
- o inter-node capabilities (e.g., "able to transfer files between this node and nodes with the following computer/OS pairs:...")

16. Responses

A standard syntax is defined for responses at the user terminal. It consists of:

<rsp-id> <rsp-text> [ON <node-id> <hst-text> <herr-code>]

The <rsp-id> is a unique identifying string for the response. There exists a command through which a user may determine details on a specified <rsp-id>.

The <node-id> is given on all errors that originate on a remote node; the <hst-text> is the host-dependent error text and is given only if it is necessary in isolating the source of the error; the <herr-code> is the host-dependent error code, also given only if necessary. There exists a command through which a user may determine details on a specified <herr-code>.

On all queries for details on a response, the information provided is static; no information is given relating to the response in the current context. In addition, if a node is given in the error response, then the node identification may be required in the user's query for details; the default node is the node that last generated an error message.

Comments

The design for response mechanisms is based on existing TCL mechanisms.

The model we have in mind for responses from remote nodes is that remote executives and procs send responses through messages to a virtual terminal process on the home node.

In regard to error information we distinguish these classes of errors:

1. Errors solely in the NOS domain, "incorrectly formatted expression" or "proc cannot be found" are examples.

2. Errors solely in the NOS domain that are caused when a user exceeds some executive configuration parameter, the number of command line continuations, for example.

3. Errors for which the description can be abstracted by the NOSCL, but for which additional host data is useful, for example, when a proc cannot be run because a host quota is exceeded.

For the second and third cases, the details on an error are node-dependent, thus node-dependent error information is included in the error response.

An alternative to giving the node and host information in the initial error response is to hide the information until the user requests it. This approach, consistent with the abstract machine approach used by Efe et al [EFE 83], is left for further study. Another alternative is to allow the user to specify whether or not host error codes are to be displayed.

Another feature of the response philosophy used by Efe is that only the highest level abstract machine generates responses to the user. All lower level machines field exceptions and abstract those exceptions according to their machine specifications. In the approach we use above, the executive or proc issues responses to the user terminal when the error is detected. Thus, while the abstract machine approach maintains an internal stack of error information, the approach we use puts the same information on the terminal.

We require that the node be specified when the user asks for details on errors originating on remote nodes. This was done for the following reasons:

- A mapping for the host-dependent error codes of all nodes should not be required at each node.

- If the error is of the second type described above, it is reasonable to assume that only the node in question has the configuration parameters for that node.

- The details on NOSCL responses—responses common to all nodes—are retained on each node along with the executive that generates the responses. This assumption is consistent with autonomy: a system administrator installs a new release of the executive and associated response files when he or she sees fit. Note that there is a natural link between a release of the executive and a release of the files containing error message details.

17. Additional issues

The following important areas have not been studied:

- How are network and machine costs modeled for the user and how does the command language accommodate the model? (See [ISO/SC16/N1217 82] for initial efforts in this area.)

- What is the context when a user designates host file specifications and host commands? What is the syntax for host commands to be executed on remote nodes? Should a dialogue with the command language interpreter on a remote host be supported?

  Particularly important is the ability for a user to bypass the NOS file system, because the NOS file system imposes at least one extra process between the file requester and the file and may therefore be too slow for the user's needs.

  Given the host file escape mechanism, the user should have the capability to determine the host name for an NOS file, and to determine the host context under which files are accessed (the default name string, privileges, and access rights).

  Note that the ability to escape into the host command language is often critical in debugging NOS functions.

- Are all NOSCL commands executed synchronously or are some asynchronous? For example, is the command that writes a record to the standard output synchronous? Note that if a procedure on node N2 is initiated from node N1, the standard output is likely to be on N1.

- How are files shared between NOS users and non-NOS users? Does the NOS have to lock other users out when it is accessing a file?

- What should be the level of support for remote batch, and how does it map into the capabilities of host systems? See the SUBMIT commands in COSCL and ANSI/X3H1 for guidance.

8.0 SUMMARY

In this paper our major concern has been how a command language changes when used in a local area net of heterogeneous computers under autonomous control.

We have concluded that the user of such a network should be exposed to the location of network resources; we identify the following key issues that must be resolved before a command language can be implemented:

- What must be done so that the network operating system efficiently supports procedures and processes executing on a node remote to the node at which they are invoked? What are the major factors in proc initiation time? . Is the number of parameters significant? Must the size of the session context be limited?

- If remote procs can be supported, are the same invocation and prompting mechanisms available as for local procs?

- Are there any restrictions on the commands available in remote procedures?

- Although costing for a distributed environment is a complex topic, the user must have some way to assess the costs resulting from a session, how are network and machine costs modeled for the user and how does the command language accommodate the model?

- Should the network operating system executives, which perform command interpretation on each node, be parameterized according to the memory and disk space available on the hosting node?

- How much of the protection mechanism must be specified in an NOS command language? How do protections map into host protection mechanisms?

- Finally, noting Thurber's advice [THURBER 81] can a useful distributed system be built on top of existing hardware and software?

9.0 REFERENCES

ABBREVIATIONS:

ANSI: American National Standards Institute

CACM: Communications of the ACM

IEEE Trans S/W Engr: IEEE Transactions on Software Engineering

IFIP: International Federation for Information Processing

ISO: International Organization for Standardization

Op Sys Rev: ACM SIGOPS Operating Systems Review

SOSP: ACM Symposium on Operating System Principles

S/W P & E: Software Practice and Experience

REFERENCES:

AGRAWALA 83
A. Agrawala, University of Maryland, personal communication with the authors

ANSI/05SD 79
ANSI, "OSCRL User Requirements," Rev 7, ANSI X3H1/05-SD, December 1979

ANSI/06SD 79
ANSI, "OSCRL Functional Requirements," Rev 5, ANSI X3H1/06-SD, December 1979

ANSI/09SD 84
ANSI, "Operating System Command and Response Language (OSCRL) Language Specification (DRAFT)," Rev 18, ANSI X3H1/09-SD, January 1984

BEECH 80
D. Beech, "What is a Command Language?" in Command Language Directions [D. Beech, ed.], North-Holland, 1980

BELL 79

Bell Laboratories, UNIX Time-sharing System: UNIX Programmer's Manual, Seventh Edition, Bell Laboratories, Incorporated, Murray Hill, New Jersey, January 1979

BIRRELL 82

A. Birrell, R. Levin, R. Needham, M. Schroeder, "Grapevine: An Exercise in Distributed Computing", Communications of the ACM, 25/4, April 1982, pp 260-274

BRADEN 80

R. Braden, N. Ludlam, "NSW Final Technical Report", UCLA TR-27, ARPANET Computing Services in Support of the National Software Works, June 1-February 1980.

BROWNBRIDGE 82

D. Brownbridge, L. Marshall, and B. Randell, "The Newcastle Connection or UNIXes of the World Unite!" S/W P & E, Vol 12, 1982, pp 1147-1162

CENTURY 83a

Century Computing, Applications Programmer's Reference Manual for the Transportable Applications Executive, Century Computing, Inc, Document No. 82-TAE-PGMV1E, November 1983

CENTURY 83b

Century Computing, User's Reference Manual for the Transportable Applications Executive, Century Computing, Inc, Document No. 82-TAE-USRV1E, November 1983

COSCL 82

CODASYL, "CODASYL Common Operating Systems Command Language (COSCL) Journal of Development," Version 2.2, CODASYL Common Operating Systems Command Language Committee, November 1982

CLARK 80

D.D. Clark and L. Svobodova "Design of Distributed Systems Supporting Local Autonomy," Spring 1980 COMPCON, pp 438-444

DAVIES 81

D.W. Davies, "Protection," Ch. 10 in Distributed Systems - Architecture and Implementation [Lampson, Paul and Siegert ed.], Springer-Verlag, 1981

DIGITAL 82a

Digital Equipment Corporation, "VAX/VMS Command Language User's Guide," Digital Equipment Corporation order no. AA-D023C-TE, May 1982

DIGITAL 82b

Digital Equipment Corporation, "DECnet-VAX User's Guide," Digital Equipment Corporation order no. AA-H802B-TE, May 1982

EFE 83

K. Efe, C. Miller, K. Hopper, "The Kiwinet-Nicola Approach: Response Generation in a User-Friendly Interface", IEEE COMPUTER, 16/9, September 1983, pp 66-78

FLETCHER 80

J. Fletcher and R. Watson, "Service Support in a Network Operating System," COMPUTER NETWORKS, Vol 4, February 1980, pp 415-424

FLETCHER 82

J. Fletcher and R. Watson, "An Overview of LINCS Architecture," UCID-19294, Lawrence Livermore National Laboratory, November 1982

FORSDICK 78

H. Forsdick, R. Schantz, R. Thomas, "Operating Systems for Computer Networks", COMPUTER, January 1978, pp 48-57.

HARDY 82

I.H. Hardy, "The Syntax of Interactive Command Languages: A Framework for Design," Software Practice and Experience, Volume 12, 1982, pp 67-75

HOARE 78

C.A.R. Hoare, "Communicating Sequential Processes," CACM 21/8, August 1978, pp 666-677

ICHBIAH 79

J.D. Ichbiah, J.C. Heliard, O. Roubine, J.G.P. Barnes, B. Krieg-Brueckner, B.A. Wichmann, "Ada Rationale," ACM SIGPLAN Notices, 14/6, June 1979

IFIP/WG2.7 1983

IFIP, "The IFIP WG 2.7 Reference Model for Command and Response Languages," [Beech and Kugler ed.], IFIP, August 1983

ISO/OSI 82

ISO, "Information Processing Systems -- Open Systems Interconnection -- Basic Reference Model," ISO/DIS 7498, April 1982

ISO/SC16/N1217 82

ISO, "OSI Management Framework (Third Working Draft)," ISO/TC97/SC16/N1217, August 1982

ISO/SC16/N1454 83

ISO, "Working Draft of File Transfer, Access and Management - The Virtual File Store," ISO/TC97/SC16/N1454, February 1983

JONES 79

A. Jones, R. Chansler Jr., I. Durham, K. Schwans, S. Vegdahl, "StarOS, a Multiprocessor System for the Support of Task Forces," Proceedings Seventh SOSP, December 1979, pp 117-127

KIMBLETON 78

S. Kimbleton, H. Wood, and M. Fitzgerald, "Network Operating Systems--An Implementation Approach", in 1978 National Computer Conference, pp 773-782

KIMBLETON 81

S. Kimbleton, P. Wang, and B. Lampson, "Applications and Protocols," Ch. 14 in Distributed Systems - Architecture and Implementation [Lampson, Paul and Siegert ed.], Springer-Verlag, 1981

KUGLER 80

H.J. Kugler, N. Lehmann, P. Putfarken, C. Unger, "The Construction of User Interfaces - A Guide for Defining Abstract Machines," Project NICOLA, University of Dortmund, November 1980

LAMPORT 78

L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," CACM 21/7, July 1978, pp 558-565

LAMPSON 81

B. Lampson, "Atomic Transactions," Ch. 11 in Distributed Systems - Architecture and Implementation [Lampson, Paul and Siegert ed.], Springer-Verlag, 1981

LANTZ 79

K. Lantz, R. Rashid, "Virtual Terminal Management in a Multiple Process Environment", Proceedings Seventh SOSP, December 1979, pp 86-97.

LANTZ 80

K. Lantz, "Uniform Interfaces for Distributed Systems", Computer Science Dept., University of Rochester, TR 63, May 1980.

LANTZ 82

K. Lantz, K. Gradischnig, J. Feldman, R. Rashid, "Rochester's Intelligent Gateway", IEEE COMPUTER, October 1982, pp 54-68

LAZOWSKA 81

E.Lazowska, H.Levy, G. Almes, M. Fischer, R. Fowler, S. Vestal, "The Architecture of the Eden System", Proceedings 8th SOSP, December 1981, pp 148-159

LISKOV 79

B. Liskov, "Primitives for Distributed Computing," Proceedings Seventh SOSP, December 1979, pp 33-42

LISKOV 81
B. Liskov, "Report on the Workshop on Fundamental Issues in Distributed Computing," ACM Op Sys Rev 15/3, July 1981, pp 9-38

MAMRAK 83
S.A. Mamrak and D. Leinbaugh, "A Progress Report on the DESPERANTO Research Project," in ACM Op Sys Rev, 17/1, January 1983, pp 17-29

OPPEN 83
D.C. Oppen and Y.K. Dalal, "The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment," in ACM Transactions on Office Information Systems, 1/3, July 1983, pp 230-253

PEEBLES 80
R. Peebles and T. Dopirak, "ADAPT: A Guest System," in IEEE Spring 1980 COMPCON, pp 445-452

POPEK 81
G. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, G. Rudisin, G. Thiel, "LOCUS: A Network Transparent, High Reliability Distributed System," Proceedings Eighth SOSP, December, 1981, pp 169-177

REED 79
D. Reed and R. Kanodia, "Synchronization with Eventcounts and Sequencers" CACM 22/2, February 1979, pp 115-123

ROBINSON 77
R. Robinson, "National Software Works: Overview and Status," IEEE COMPCON, Fall 1977

ROWE 82
L. Rowe and K. Birman, "A Local Network Based on the UNIX Operating System", IEEE Trans S/W Engr, pp 137-146, Volume SE-8, March 1982.

SALTZER 78
J. Saltzer, "Research Problems of Decentralized Systems with Largely Autonomous Nodes," ACM Op Sys

Rev, Volume 12, January 1978, pp 43-52

SCHICKER 75
P. Schicker, W. Baechi, and A. Duenki, "Job Control in a Heterogeneous Computer Network", ONLINE-75 Conference, Online Conference Limited 1975, Uxbridge, England, pp 537-545

SCHNEIDER 82
M.L. Schneider, "Models for the Design of Static Software Assistance" in Directions in Human Computer Interaction [ed Shneiderman and Badre], Ablex, 1982

SHNEIDERMAN 80
B. Shneiderman, Software Psychology, Winthrop Publishers, Cambridge, 1980

SMITH 82
D.C. Smith, C. Irby, R. Kimball, B. Verplank, "Design of the STAR User Interface," BYTE, April 1982, pp 242-282

THURBER 81
K. Thurber, page 491 in Distributed Systems - Architecture and Implementation [Lampson, Paul and Siegert ed.], Springer-Verlag, 1981

WATSON 81
R. Watson "Identifiers (naming) in Distributed Systems," Ch. 9 in Distributed Systems - Architecture and Implementation [Lampson, Paul and Siegert ed.], Springer-Verlag, 1981

APPENDIX A

CHECKLIST FOR NETWORK COMMAND LANGUAGES

The following is a list of questions to use in examining a given network command language.

- o Architecture of the net:
  - Local net or long haul?
  - Heterogeneous or homogeneous?
  - If homogeneous, what machine/OS constitutes the nodes?
  - If heterogeneous, what machines/OS's supported?
  - Gateways supported?
  - Autonomous nodes?
- o Purpose of the net: Is the net used in a production environment? A research environment?
- o General description of the command language
  - Did the command language antedate the network? If so, was it changed to accomodate the network?
  - Any commands that would not exist if the N/W did not exist?
  - Does the command language look the exactly same from any terminal on any node?

CHECKLIST FOR NETWORK COMMAND LANGUAGES

- Are procedures supported? Are remote procedures supported? Any limitations on what goes in them? (e.g., can a remote procedure read from the user's terminal?)

- Are remote logins supported?

- Is there a uniform method for logging in to any terminal on any node?

o File system

- Are names hierarchical or flat?

- Central or distributed directory?

- Filespec?

- Protections? (e.g., capability? access lists?)

- Replicated directory?

- Can all the resources (i.e., files and devices) required by a proc can be secured before the proc is run? If so, how? (E.g., user-explicit lock)

- Is remote record access supported? If not, are files copied upon proc initiation?

o Performance:

- Time to locate and initiate a proc?

- Time to copy a null file between two nodes?

- Time to copy an n-byte text file between two nodes assuming no translation?

- Time to copy an n-byte text file between two nodes assuming translation (e.g., EBCDIC to ASCII)?

- Time to copy an n-byte data file assuming data type translation?

CHECKLIST FOR NETWORK COMMAND LANGUAGES

- Interrupts:

  - Is the attention sequence terminal and node-independent?
  - How is a remote proc aborted?
  - Can a user run more than one proc at a time? If not, can a remote proc be interrupted while some built-in commands (e.g., "show status" or "help") are executed?

- How does a user get charged for:

  - Execution of a remote proc
  - Use of a file, directory, or device on a remote system (including execution of a remote file server)
  - Maintenance of remote files
  - Remote mounting of tapes

  Is there some model of a user getting charged to the one user id under which the user logs on (i.e., the user's id under the network OS)? If so, how is this accumulated?

- Privileges:

  - Any notion of privileges in the user view?
  - What kind of privileges does the user have under the host?

- Quotas/limits:

  - Any notion of quotas in the user view?
  - What kind of quotas does the user have under the host?

- Crash/recovery:

  - Is the user informed of the crash of a remote node?

  - Does the NOS attempt any recovery?

- o Miscellaneous:

  - Special security mechanisms (e.g., extra passwords/privileges)?

  - Any special resource management considerations? (TBD)

  - Can a user send a message to anyone on the net using the same command and addressing as a local message? Can the user send a message to a system operator?

END

DATE

FILMED

SEP 27 1984

End of Document